

**ДИПЛОМНА РОБОТА
БАКАЛАВРА**

ДРБ. ІІ - 29.00.00.000 ІІЗ

Група ІІ-22-1К

Борис Максим

2024

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

УДК 004.4

ДИПЛОМНА РОБОТА БАКАЛАВРА

Тема: “Реалізація програмного забезпечення системи графічного
моделювання”

(назва згідно з наказом ректора)

спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДРБ.ІІ - 29.00.00.000 ПЗ
(позначення)

Консультанти:

Студент

ІІ-22-1К /М.А.Борис/
(шифр групи) (підпис) (дата) (розшифровка підпису)

Керівник роботи

к.т.н., доц. /Р.Б.Вовк/
(посада) (підпис) (дата) (розшифровка підпису)

з питань нормоконтролю

Допускається до захисту
Завідувач кафедри

асист. /М.М. Піх/
(посада) (підпис) (дата) (розшифровка підпису)

к.т.н., доц. /В.В.Бандура/
(посада) (підпис) (дата) (розшифровка підпису)

2024
(рік)

Івано-Франківський національний технічний університет нафти і газу
Інститут, факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Освітньо-кваліфікаційний рівень бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедрою ІІЗ

доц. В.В. Бандура

“ ” 2024 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Борису Максиму Андрійовичу

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) “Реалізація програмного забезпечення системи графічного моделювання”

керівник проекту (роботи) Вовк Роман Богданович, доцент

затверджені наказом закладу вищої освіти від “ ” травня 2024 р. №

2. Строк подання студентом проекту (роботи) 17 червня 2024 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області використання ІТ для графічного моделювання та рендерингу

2. Моделювання та розробка структури системи графічного моделювання

3. Розробка та проектування структури бази даних

4. Розробка архітектури та програмна реалізація системи графічного моделювання

5. Виконання розгортання програмних сервісів

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1. Тривимірне сітчасте уявлення тривимірного об'єкту (рис. 1.1)

2. Типовий конвеєр рендеринга (рис. 1.2)

3. Збірка примітивів (рис. 1.3)

4. Об'єкти, що виходять за межі оглядового вікна (рис. 1.4)

5. Етап растеризації, на якому примітиви перетворюються в фрагменти (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 22 травня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області використання ІТ для графічного моделювання та рендерингу	26.05.2024	виконано
2	Моделювання та розробка структури системи графічного моделювання	29.05.2024	виконано
3	Розробка та проектування структури бази даних	30.05.2024	виконано
4	Розробка архітектури та програмна реалізація системи графічного моделювання	04.06.2024	виконано
5	Виконання розгортання програмних сервісів	10.06.2024	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	17.06.2024	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

В даній роботі представлено розробку програмного забезпечення для системи графічного моделювання та рендерингу у веб-середовищі. Метою проекту є створення інтерактивної платформи, яка дозволяє користувачам створювати, редагувати та візуалізувати тривимірні моделі без потреби встановлення спеціалізованого програмного забезпечення на локальному комп'ютері. Основні завдання включають розробку інтуїтивно зрозумілого інтерфейсу користувача, інтеграцію з популярними бібліотеками для рендерингу графіки, забезпечення високої продуктивності та якості рендерингу, а також впровадження засобів для обміну та співпраці між користувачами.

КЛЮЧОВІ СЛОВА: ГРАФІЧНЕ МОДЕЛЮВАННЯ, РЕНДЕРИНГ, ОБ'ЄМНА МОДЕЛЬ, БАЗА ДАНИХ, МІКРОСЕРВІСНА АРХІТЕКТУРА, ТРИВИМІРНІ МОДЕЛІ.

ANNOTATION

This work presents the development of software for a system of graphic modeling and rendering in the web environment. The goal of the project is to create an interactive platform that allows users to create, edit and visualize three-dimensional models without the need to install specialized software on a local computer. Key tasks include developing an intuitive user interface, integrating with popular graphics rendering libraries, ensuring high performance and rendering quality, and implementing tools for user sharing and collaboration.

KEY WORDS: GRAPHIC MODELING, RENDERING, VOLUME MODEL, DATA BASE, MICROSERVICE ARCHITECTURE, THREE-D MODELS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	8
ВСТУП.....	9
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ГРАФІЧНОГО МОДЕЛЮВАННЯ ТА РЕНДЕРИНГУ	11
1.1. Сітчасті зображення графічних об'єктів та візуалізація графічних об'єктів.....	11
1.2. Поняття освітленості графічної сцени	14
1.3. Аналіз алгоритмів розрахунку освітленості графічних сцен.....	18
1.3.1. Алгоритм Forward рендерінгу.....	18
1.3.2. Алгоритм Deferred рендерінгу	20
1.3.3. Алгоритм Forward+ рендерінгу	23
1.4. Огляд платформ геометричного моделювання та рендерінгу зображень	25
1.4.1. Платформа рендерінгу VTK Reader.....	26
1.4.2. Програма інтерактивної наукової візуалізації ParaView.....	27
1.4.3. Аналіз EEVEE render	29
1.4.4. Аналіз Corona render	32
1.4.5. Веб-платформа комп'ютерної графіки Shadertoy	33
2. МОДЕЛЮВАННЯ ТА РОЗРОБКА СТРУКТУРИ СИСТЕМИ ГРАФІЧНОГО МОДЕЛЮВАННЯ.....	36
2.1. Аналіз вимог до розроблюваного програмного забезпечення	36
2.2. Розробка бізнес моделі	40

					ДРБ.ПІ – 29.00.00.000 ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Борис М.А.			Реалізація програмного забезпечення системи графічного моделювання Пояснювальна записка	Літ.	Арк.	Акрушів
Перевір.		Вовк Р.Б.					6	
Реценз.						ІФНТУНГ ІІ-22-1К		
Н. Контр.		Піх М.М.						
Затверд.		Бандура В.В.						

2.3. Розробка та проектування структури бази даних.....	43
2.4. Реалізація алгоритмів та функціоналу процесів реєстрації та авторизації.....	47
3. РОЗРОБКА АРХІТЕКТУРИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ГРАФІЧНОГО МОДЕЛЮВАННЯ.....	60
3.1. Опис архітектури сервісу замовлень.....	60
3.2. Конструювання системи графічного моделювання згідно принципів REST	63
3.3. Розробка інтерфейсу користувача системи графічного моделювання та рендерингу.....	66
3.4. Виконання розгортання програмних сервісів.....	70
ВИСНОВКИ	74
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	75
ДОДАТКИ	
БІБЛІОГРАФІЧНА ДОВІДКА	

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CTS – Common Type System, система спільних типів.
CLR – Common Language Runtime, загальномовне середовище виконання.
IL – Intermediate Language, проміжна мова.
GC – Garbage Collector, збирач сміття.
DI – Dependency Injection, ін'єкція залежностей.
SSE – Server-sent Events, події, що надіслані сервером.
JWT - JSON Web Token, токен у форматі файлу JSON
DAL – Data Access Layer, шар доступу до даних.
BLL – Business Logic Layer, шар бізнес-логіки.
PL – Presentation Layer, шар представлення.
CUDA - Compute Unified Device Architecture
IBR - Візуалізація на основі зображення
JPEG - Joint Photographic Experts Group
LOD - Рівень деталізації
MPEG - Moving Picture Experts Group
RFB - Віддалений буфер кадру
RTP - Real-time Transport Protocol
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
VNC - Virtual Network Computing
VTK - Набір інструментів візуалізації

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

ВСТУП

3D графіка демонструє експоненціальне зростання популярності. Триваючий прогрес в галузі графіки в реальному часі останнім часом привів до появи численних мультимедійних додатків в реальному часі. Існують чотири основні проблеми щодо графічних додатків для хмарних обчислень:

- Затримка: час, необхідний для передачі графічного контенту з сервера клієнту;
- Надійність: дані часто губляться або пошкоджуються;
- Пропускна здатність: скільки даних може бути відправлено за цей час;
- Пропускна здатність пристроїв кінцевого користувача: можливість обробки графіки. Необхідно враховувати всі ці міркування, а також компроміси.

Рендеринг 3D графіки передбачає величезні вимоги до кінцевого користувача. Існує різні підходи до вирішення цієї проблеми. Одним з варіантів є створення сервісу обробки даних на стороні сервера. Хоча на перший погляд ця пропозиція є багатообіцяючою, існує багато компромісів, які необхідно обов'язково вивчити, щоб реалізувати переваги хмарного сервісу для рендерингу 3D графіки. Передача графічного контенту, особливо більших 3D моделей, через мережі може постраждати від задержок передачі та низької пропускну здатності. Крім того, обробка графічних даних вважається тривалим процесом, і в разі, якщо серверу доведеться обслуговувати велику кількість клієнтів, він може виявитися перевантаженим; робоче навантаження рендеринга може бути перенесена на клієнта, щоб скористатися його можливостями обробки графіки і звільнити сервер від обчислювального навантаження. Однак це призведе до подальшого збільшення обсягу роботи, виконуваної клієнтом. З точки зору мережі, повинна існувати достатня пропускна здатність для підтримки більш

					ДРБ.ІІ – 29.00.00.000 ІІЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

великих передач даних між клієнтом і хмарою. Крім того, мережа повинна буде мати досить низьку затримку для підтримки послуг і інтерактивності в режимі реального часу. Сервери також вимагають достатньої пропускної спроможності для своєчасної обробки 3D-рендеринга.

Для використання об'ємних моделей недостатньо лише їх створити. Необхідно також мати можливість представити створену модель для кінцевого користувача. Одним із способів представлення моделей є рендеринг.

Даний процес дозволяє представити модель у вигляді зображення. Алгоритми рендерингу відрізняються один від одного швидкістю та якістю отриманого результату.

Моделі мають тенденцію ставати деталізованішими, алгоритми – ресурсовимогливішими, а набір вимог до кінцевого результату – все більшим. Для отримання результату, що відповідає сформульованому набору вимог, необхідні досить потужні обчислювальні засоби, оскільки алгоритми рендерингу є вкрай ресурсовитратними. Навіть найдорожчі персональні комп'ютери можуть досить швидко перестати відповідати потребам, а заміна комплектуючих є особливо дорогавартісною в наш час у зв'язку зі світовим дефіцитом напівпровідників, які необхідні для виготовлення комп'ютерних комплектуючих.

Для подолання цієї проблеми було вирішено створити програмні засоби, що надаватимуть можливість виконати рендеринг з будь-якого пристрою з доступом до мережі Інтернет.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ГРАФІЧНОГО МОДЕЛЮВАННЯ ТА РЕНДЕРИНГУ

1.1. Сітчасті зображення графічних об'єктів та візуалізація графічних об'єктів

Тривимірні (3D) графічні об'єкти можуть бути представлені як безліч багатокутників або так звана багатокутна сітка. Розглянемо тільки трикутну сітку, оскільки вона є одним з найбільш поширених уявлень 3D-об'єктів. Будь-який не-трикутний багатокутник може бути триангульований після декількох простих кроків.

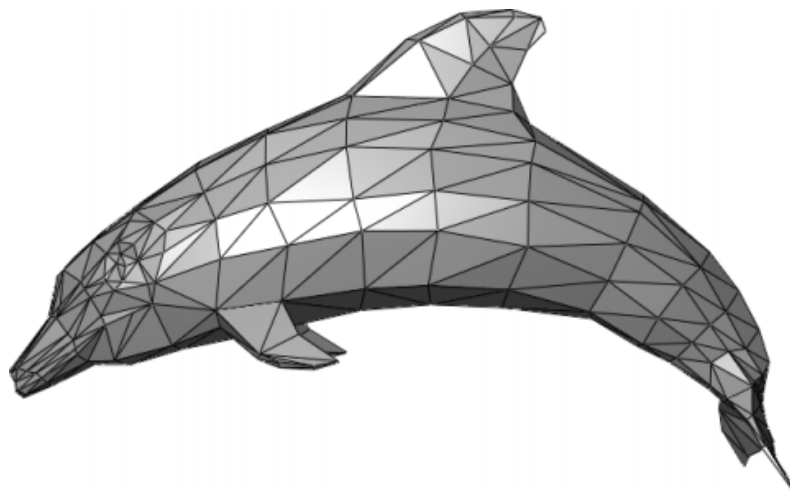


Рисунок 1.1 - Тривимірне сітчасте уявлення тривимірного об'єкту

Існує три типи інформації для подання тривимірної сітки, включаючи геометричну інформацію, інформацію про зв'язок і фотометричну інформацію. Ми можемо уявити сітку як $M = (V, F)$, де V - список вершин ($v_1, v_2, \dots, v_n$), а F - список трикутників ($tri_1, tri_2, \dots, tri_n$). Кожна верхова координата може бути виражена трьома значеннями з рухомою комою (x, y, z), і кожен трикутник (або грань) виражається трьома цілими числами, що відносяться до трьох вершин, які утворюють трикутник.

Графічний рендеринг - це процес одночасної генерації 2D зображень з 3D-сцен. Графічні дані перед відображенням на екрані повинні пройти кілька етапів в так званому конвеєрі графічного рендерингу. Графічний конвеєр зазвичай складається з декількох етапів, включаючи обробку вершин, примітивну збірку, геометричну обробку, обрізання і відбір, растеризування і обробку фрагментів.

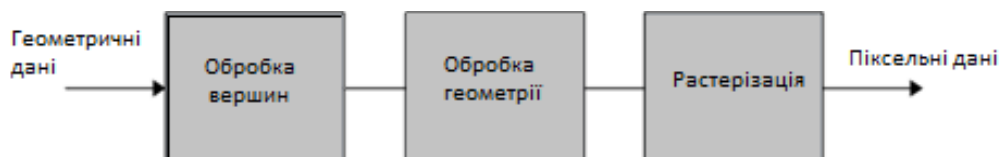


Рисунок 1.2 - Типовий конвеєр рендеринга

Вершинні шейдери відповідають за обробку вершин шляхом виконання таких операцій, як перетворення вершин, висвітлення освітлення. Виходами на цьому етапі є окремі версії.

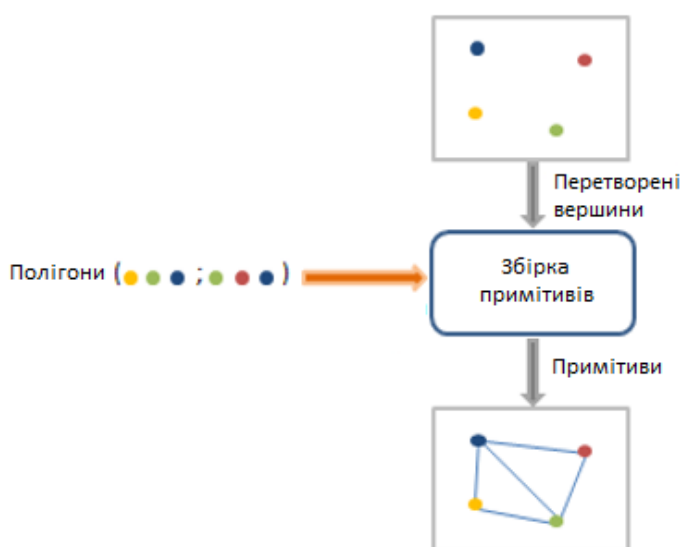


Рисунок 1.3 - Збірка примітивів

Збірка примітивів: на цьому етапі перетворені вершини групуються на основі інформації про зв'язок, який повинен бути перетворено в примітиви (багатокутники, прямі, точки).

Геометрична обробка: Стадія обробки геометрії відбувається до відбору / обрізки і після примітивної збірки. Він отримує примітиви від попереднього етапу для подальшої обробки. На відміну від інших етапів, геометрична стадія здатна генерувати нові примітиви з існуючих.

Відсікання і відбір: Цей етап відповідає за усунення невидимих примітивів і тих, які випадають за межі перегляду.

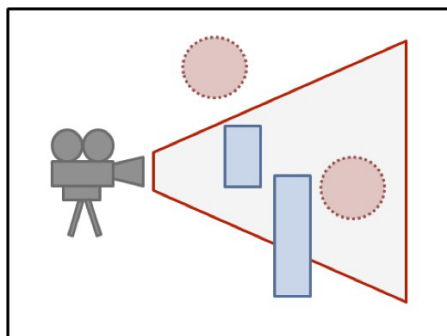


Рисунок 1.4 - Об'єкти, що виходять за межі оглядового вікна

Растрезація: стадія растрезації відповідає за перетворення кожного примітива, в набір фрагментів. Результати цього етапу будуть передані на етап обробки фрагментів для подальшої обробки.

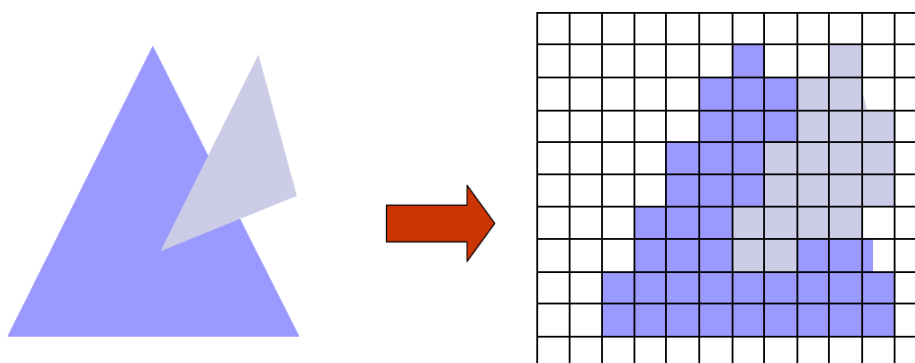


Рисунок 1.5 - Етап растрезації, на якому примітиви перетворюються в фрагменти

Стадія обробки фрагментів: Вихід фрагмента зі стадії растрезації дорівнює розміру пікселя, але не є реальним пікселем. При обробці

фрагментів дані повинні пройти ряд тестів (наприклад, тест на глибину, тест на трафарет) і стати реальним пікселем, який буде відображатися на екрані після проходження всіх цих тестів.

1.2. Поняття освітленості графічної сцени

Кожен елемент графічної сцени має дві властивості: геометричну форму та те, як поверхня даного об'єкту відображається.

Найпоширенішим представленням об'єктів графічної сцени є використання полігонів для формування об'єму об'єктів сцени. В якості полігонів використовуються багатокутники з найменшою кількістю кутів, що можуть мати площу – тобто, трикутники.

Знаючи геометричну форму об'єкта, можна визначити, яку частину екрану займає даний об'єкт, та чи потрапляє він на екран взагалі. У випадку, якщо об'єкт потрапляє на екран, потрібно визначити, як даний об'єкт в даній точці буде відображений.

Зважаючи той факт, що сучасні дисплеї складаються з пікселів, які, в свою чергу, складаються з субпікселів, кожен з яких може відображати червоний, зелений або синій колір, то проблема відображення об'єкту зводиться до визначення кольору, який буде мати екран в певній точці.

Існує багато різних підходів до визначення результуючого кольору об'єкту в певній точці. Найпростішим є задання базового кольору для всього об'єкту і використання його для кожного пікселя, на який потрапляє об'єкт графічної сцени. Такий метод зображено на рисунку 1.6. Модифікацією даного методу визначення кута між нормаллю поверхні та вектором камери. Даний метод зображено на рисунку 1.7.

Більш складним методом визначення кольору об'єкту є запровадження джерел освітлення. Даний метод відображення об'єктів зображено на рисунку 1.8.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

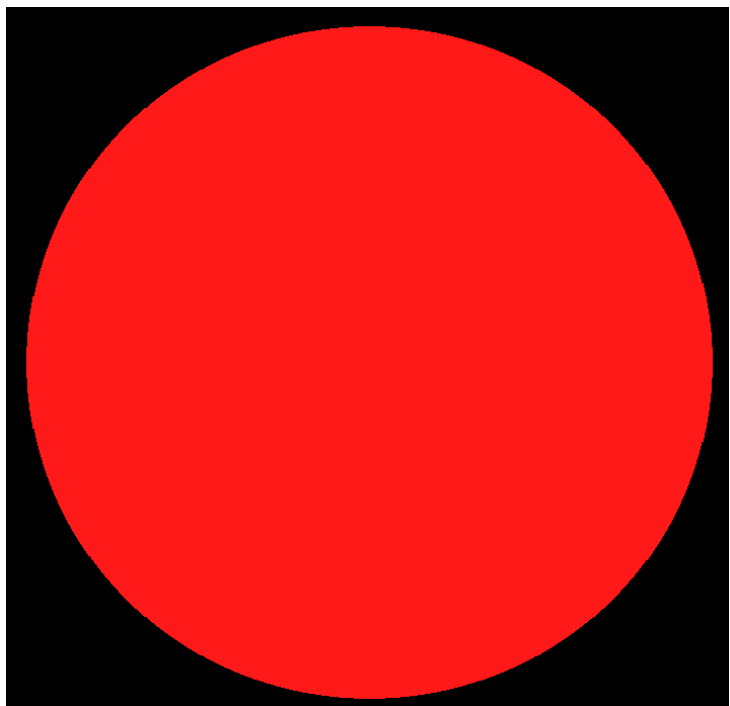


Рисунок 1.6 – Відображення об’єкту графічної сцени з використанням базового кольору

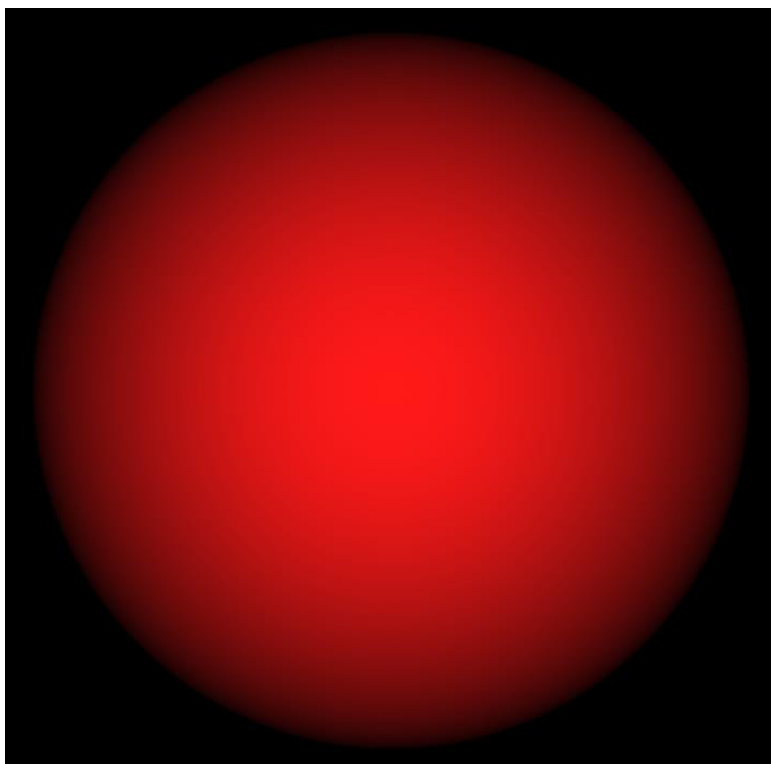


Рисунок 1.7 – Відображення об’єкту графічної сцени з визначенням кута між нормаллю поверхні та вектором камери

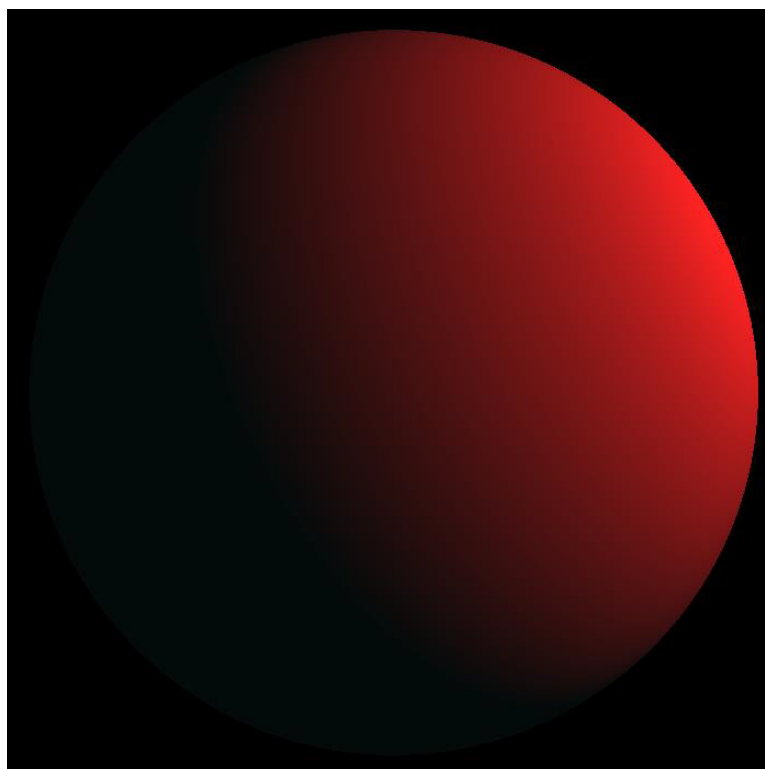


Рисунок 1.8 – Відображення об’єкту графічної сцени з використанням одного джерела освітлення

З розвитком комп’ютерних технологій з’явилася потреба рендерити об’єкти з максимальним наближенням до реальності. Даний підхід до генерації зображень отримав назву фізично-коректного рендерінгу.

Суть даного підходу полягає в дотриманні законів фізики, зокрема закону збереження енергії, та використанні моделей відображення об’єктів, що використовують реально існуючі фізичні властивості матеріалів.

Вважається, що фізично коректний рендерінг базується на моделях, що засновані на понятті мікрогеометрії. Кожна видима точка поверхні складається з багатьох нормалей мікроповерхонь, які відбивають світло в різні напрямки. Зважаючи на те, що орієнтація мікроповерхонь є випадковою, то їх розподіл представляється статистично. Для більшості поверхонь розподіл мікроповерхонь є неперервним, з піком, що збігається з нормаллю макроповерхні. «Ширина» даного розподілу визначається шорсткістю поверхні. Чим поверхня більш шорстка, тим «ширше» буде

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		16

розподіл, що означає, що нормалі мікроповерхонь направлені в різні сторони. Приклад освітлення за допомогою фізично коректної моделі зображено на рисунку 1.9.

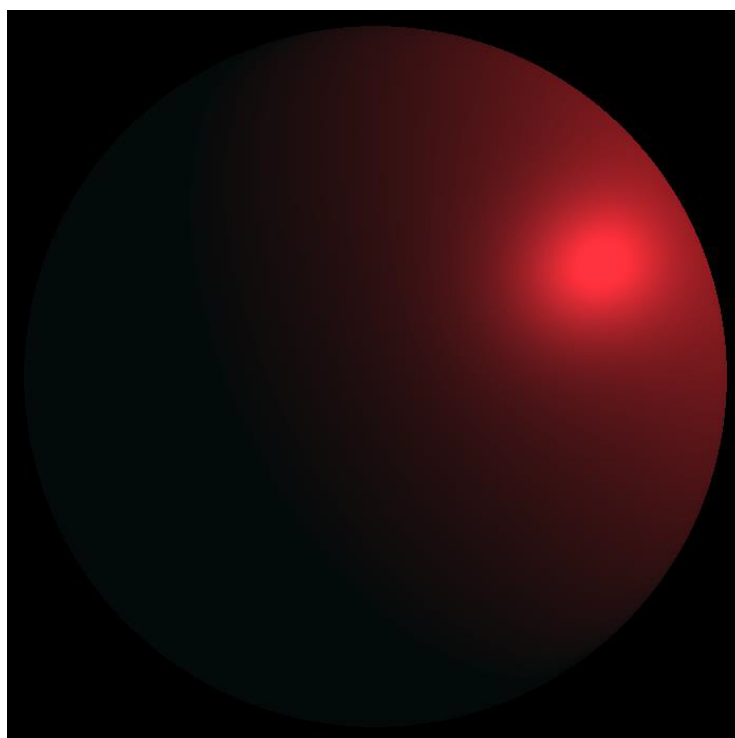


Рисунок 1.9 – Освітлення об’єкту за допомогою фізично-коректної моделі

Використання фізично-коректних моделей освітлення дозволяє наблизити результуюче зображення до реальності. Але обчислення за допомогою методу фізично-коректного освітлення з точки зору обчислень набагато важче, ніж будь-яким іншим, представленим вище, методом.

Порівнявши швидкодії різних методів освітлення об’єктів видно, що використання фізично-коректних методів освітлення накладає певні обмеження. Головним обмеженням є кількість джерел освітлення на кадр. Дане обмеження є неприємним, адже художники по освітленню, що займаються налаштуванням освітлення графічних сцен, мають значно менше можливостей, аніж якби даного обмеження не існувало.

Саме тому актуальним постає питання використання ефективних алгоритмів відображення об’єктів.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

1.3. Аналіз алгоритмів розрахунку освітленості графічних сцен

Історично найбільш поширеними алгоритмами розрахунку освітленості графічних сцен є наступні: Forward, Deferred, Forward+. Кожен наступний алгоритм має переваги відносно попереднього. Так, найпершим і найпростішим алгоритмом є Forward рендерінг.

1.3.1. Алгоритм Forward рендерінгу

Алгоритм Forward рендерінгу можна описати наступним чином: для кожного об'єкту сцени розрахувати освітленість відносно кожного джерела освітлення.

Окрема шейдерна програма буде відповідати повністю за підрисовування об'єкта, починаючи з вершинного шейдера і до фрагментного шейдера. Джерела освітлення, в такому випадку, представляються аналітично: місце в просторі та описом обмежуючого простору, в якому дане джерело освітлення знаходиться.

Таким чином, в фрагментному шейдері кожного об'єкту будуть проводитись розрахунки з кожним джерелом освітлення графічної сцени, чи потрапляє дане джерело освітлення на певну точку даного об'єкту.

Результатом фрагментного шейдеру буде результуюче зображення освітленості графічної сцени. Послідовність операцій Forward рендерінгу зображено на рисунку 1.10.

Але зі збільшенням кількості джерел освітлення, збільшується і час виконання Forward рендерінгу, що є його головним недоліком.

Ще одним недоліком алгоритму Forward рендерінгу є накладання різних об'єктів графічної сцени на одні й ті самі пікселі екрану. Це відбувається в тому випадку, коли два об'єкти графічної сцени розташовані на різній відстані від камери, але вздовж умовної прямої. Спочатку відбувається розрахунок обчислення освітленості для одного об'єкту, а потім

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

це значення відкидається, і відбувається розрахунок освітленості для того ж самого пікселя, але іншого об'єкту сцени.

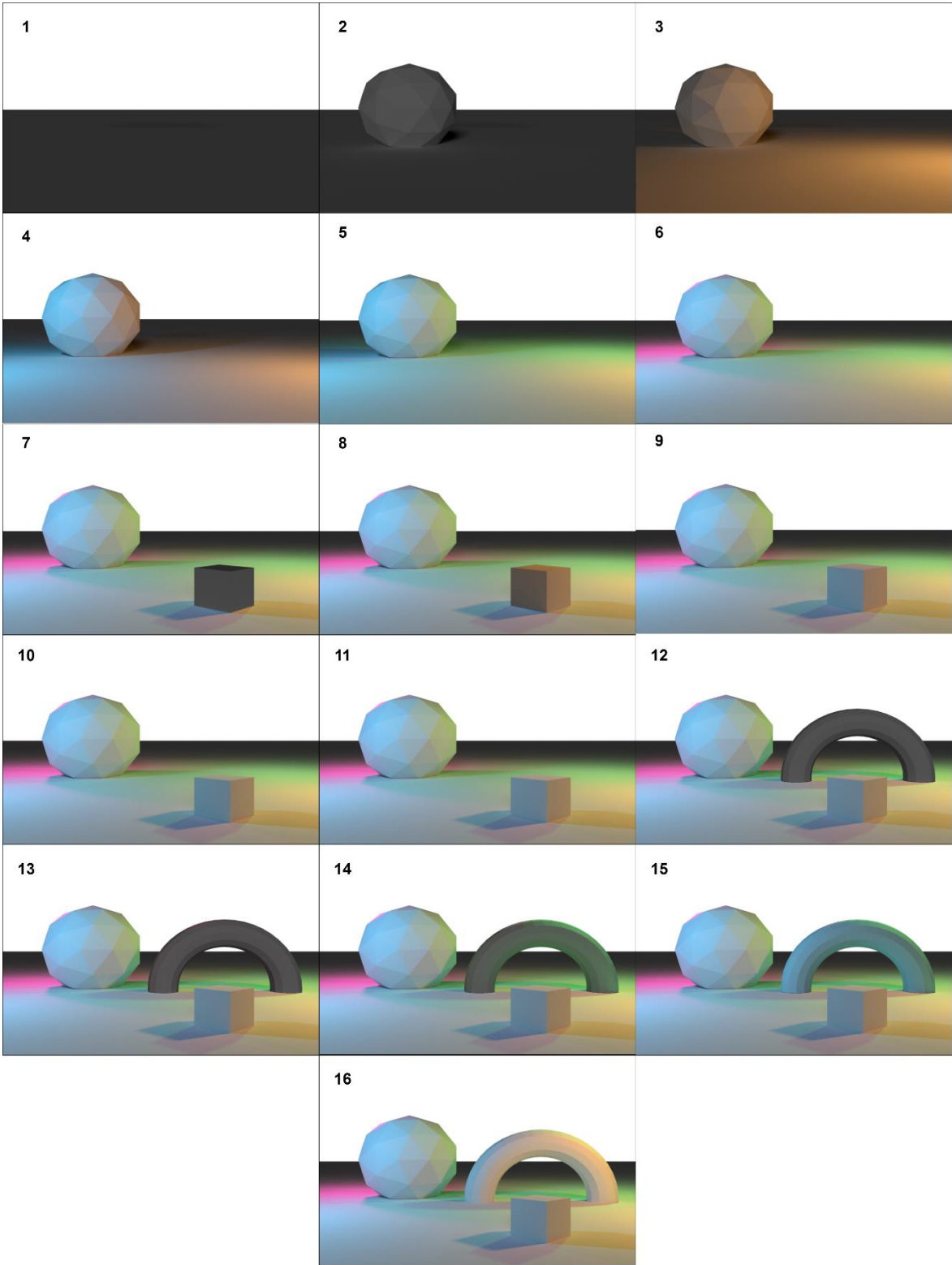


Рисунок 1.10 – Послідовність операцій Forward рендерінгу

Блок-схема алгоритму Forward рендерінгу зображена на рисунку 1.11.



Рисунок 1.11 – Блок-схема алгоритму Forward рендерінгу

1.3.2. Алгоритм Deferred рендерінгу

Алгоритм Deferred рендерінгу має дві ключові відмінності від Forward рендерінгу.

Першою відмінністю є те, що результатом фрагментного шейдеру є набір значень, таких як: шорсткість (roughness), металевість (metalness), базовий колір (base color), нормаль поверхні (normal), світимість поверхні (emissive) та глибина об'єкту відносно камери (depth).

Дані значення записуються в окремі текстури. Ці текстури називаються GBuffer. Приклад GBuffer зображено на рисунку 1.12.

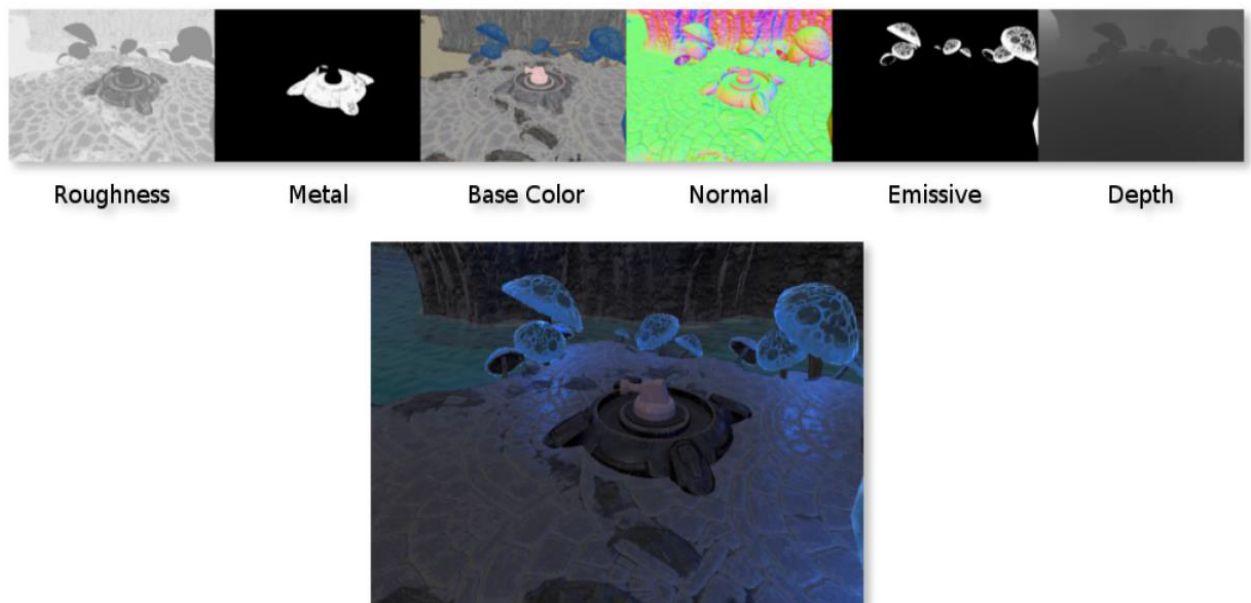


Рисунок 1.12 - Вивід зображення в декілька буферів

Після формування GBuffer графічної сцени, відбувається обчислення освітленості графічної сцени з використанням значень з текстур GBuffer.

Другою відмінністю алгоритму Deferred рендерінгу є те, що джерело освітлення в даному випадку представляється у вигляді меша, який рендериться як звичайний об'єкт графічної сцени. Для кожного пікселя, на який потрапляє даний меш, відбувається розрахунок освітленості.

Дана оптимізація дозволяє прибрати розрахунок освітленості для тих пікселів, на які не потрапляють джерела освітлення. Покрокова робота алгоритму Deferred рендерінгу зображена на рисунку 1.13.

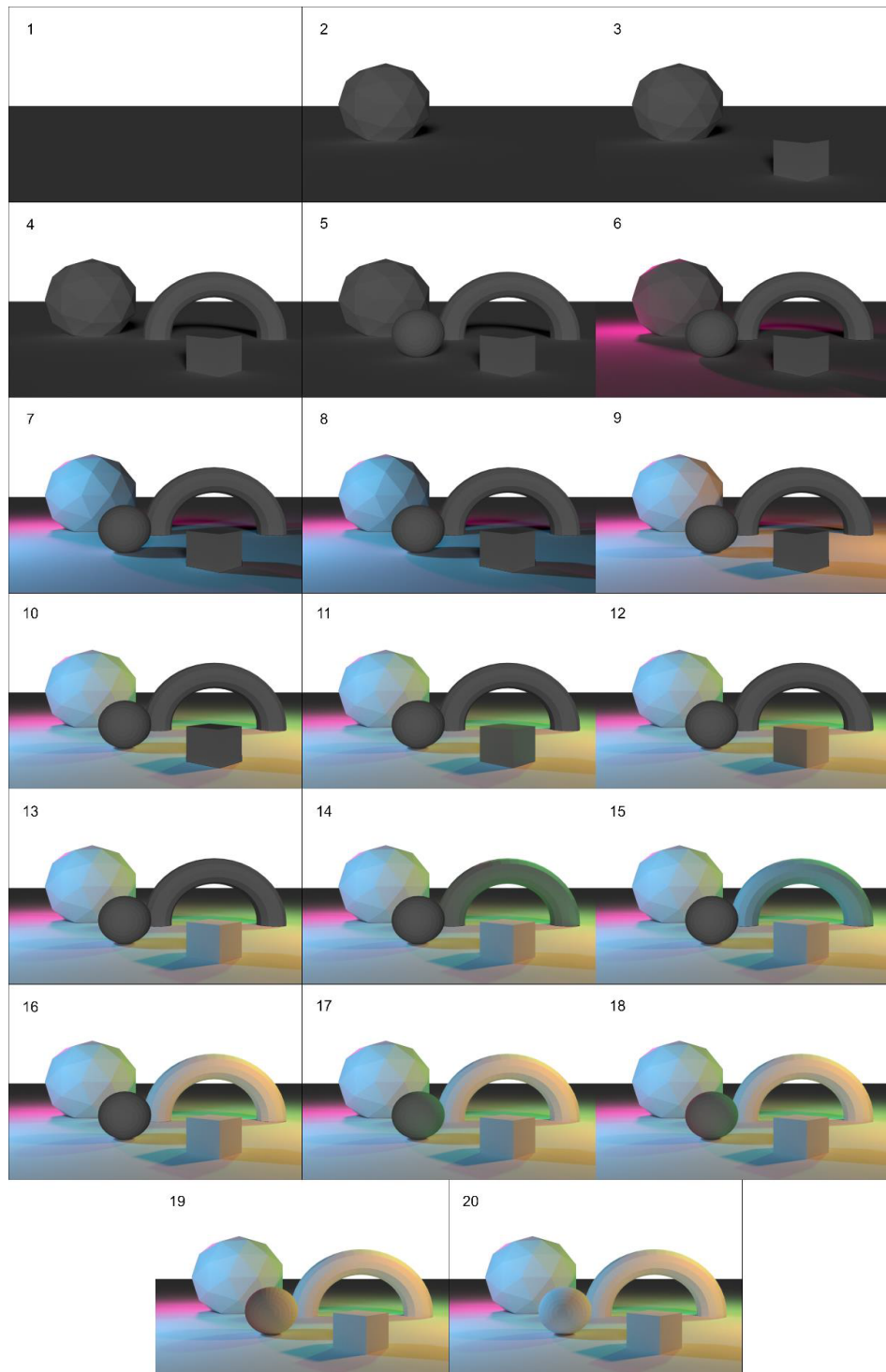


Рисунок 1.13 – Покрокова робота алгоритму Deferred рендерінгу

Також, даний алгоритм вирішує проблеми накладання різних об'єктів графічної сцени на один і той самий піксель екрану. Тобто, для кожного пікселя розрахунок освітленості відбувається лише один раз. Блок-схема роботи алгоритму Deferred рендерінгу зображений на рисунку 1.14.

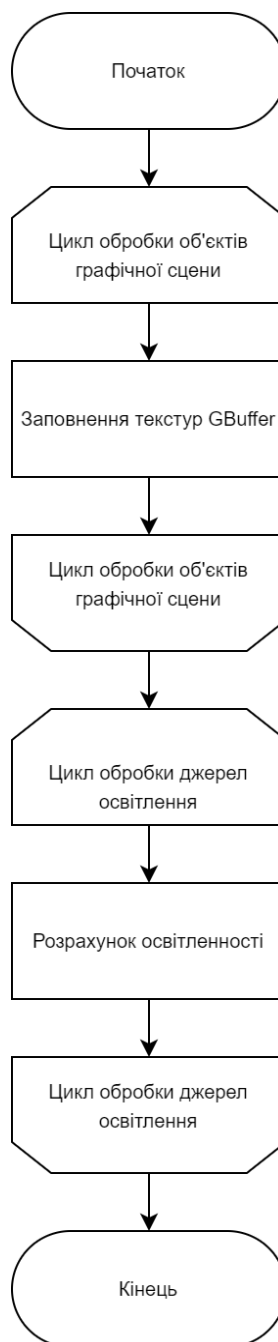


Рисунок 1.14 – Блок-схема алгоритму Deferred рендерінгу

1.3.3. Алгоритм Forward+ рендерінгу

Алгоритм Forward+ рендерінгу є хронологічно наймолодшим серед трьох. Поява даного алгоритму тісно пов'язана з графічним API DirectX 11, в рамках якого було представлено новий тип шейдерів – compute шейдер. Це дозволило використовувати графічні процесори для обчислень загального

призначення. Саме compute шейдери використовуються в алгоритму Forward+ рендерінгу.

Ідея Forward+ рендерінгу наступна. Екран розбивається на множину тайлів та створюється спеціальний буфер, в якому зберігаються, які з джерел освітлення потрапляють на певний тайл екрану. Далі під час обчислень освітленості графічної сцени до уваги беруться лише ті джерела освітлення, що знаходяться у відповідному буфері того тайлу, до якого належить відповідний піксель. Послідовність фільтрації джерел освітлення зображена на рисунку 1.15.

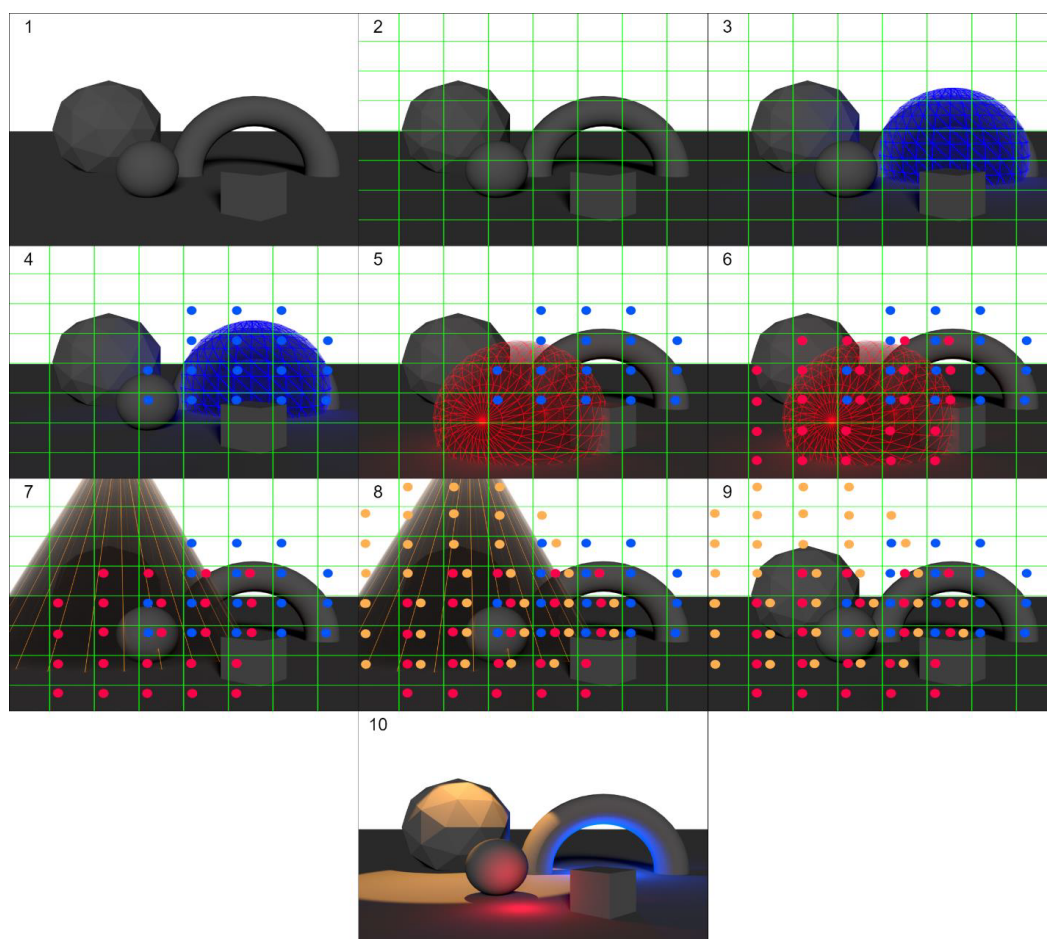


Рисунок 1.15 – Покрокова робота алгоритму фільтрації джерел освітлення

Алгоритм Forward+ рендерінгу також схожий на алгоритм Deferred рендерінгу тим, що спочатку формується GBuffer. Потім відбувається фільтрація джерел освітлення для кожного тайлу екрану. Останнім етапом

алгоритму є обчислення освітленості для кожного пікселя тайлу, де до уваги беруться лише ті джерела освітлення, що відносяться до даного тайлу. Блок-схема алгоритму Forward+ рендерінгу зображена на рисунку 1.16.

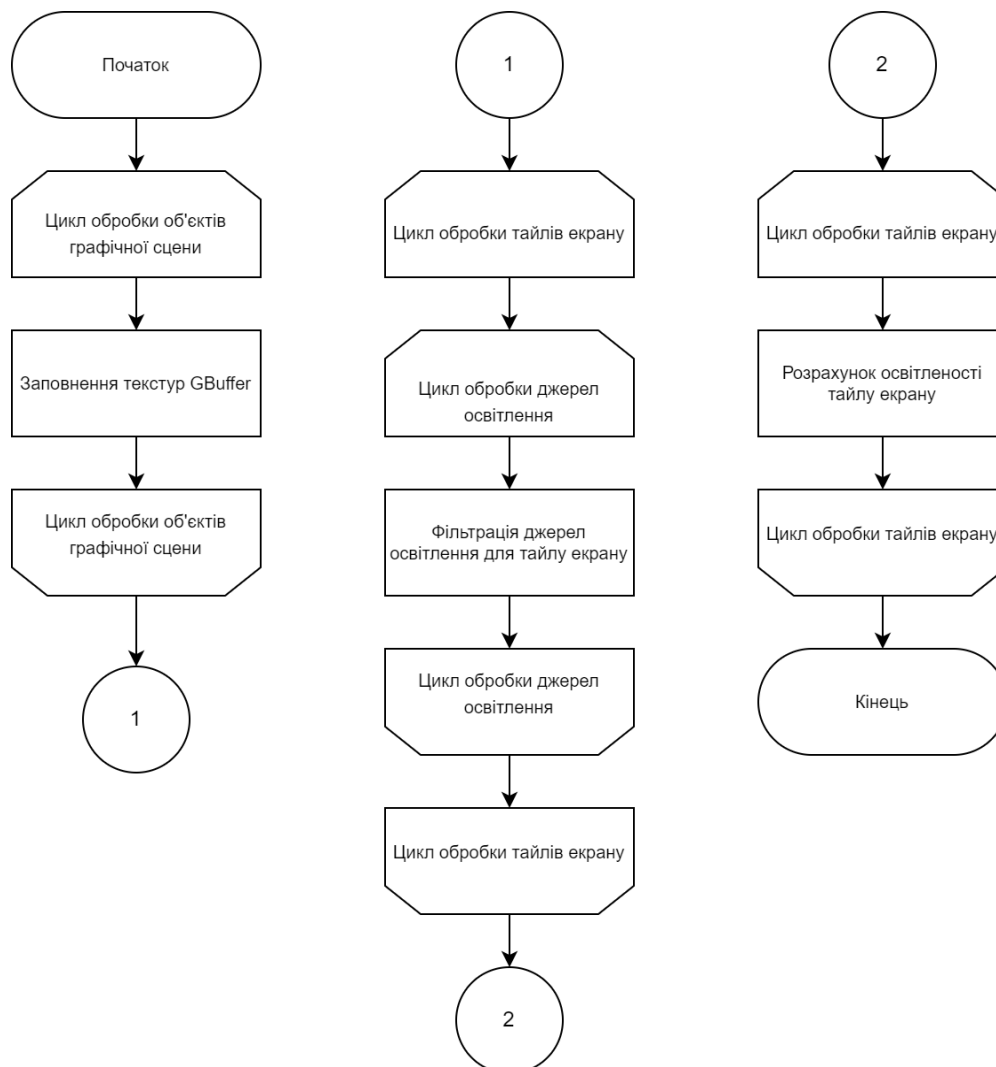


Рисунок 1.16 – Блок-схема алгоритму Forward+ рендерінгу

1.4. Огляд платформ геометричного моделювання та рендерінгу зображень

За десятки років існування VTK (набір інструментів візуалізації) було створено чимало програм для рендерінгу зображення, серед них:

- VisIt;
- VtkReader;

- Paraview.

1.4.1. Платформа рендерингу VTK Reader

Більшість з цих програм мають інтуїтивно зрозумілий користувальницький інтерфейс, наприклад ось так виглядає рендеринг куба у VTK Reader.

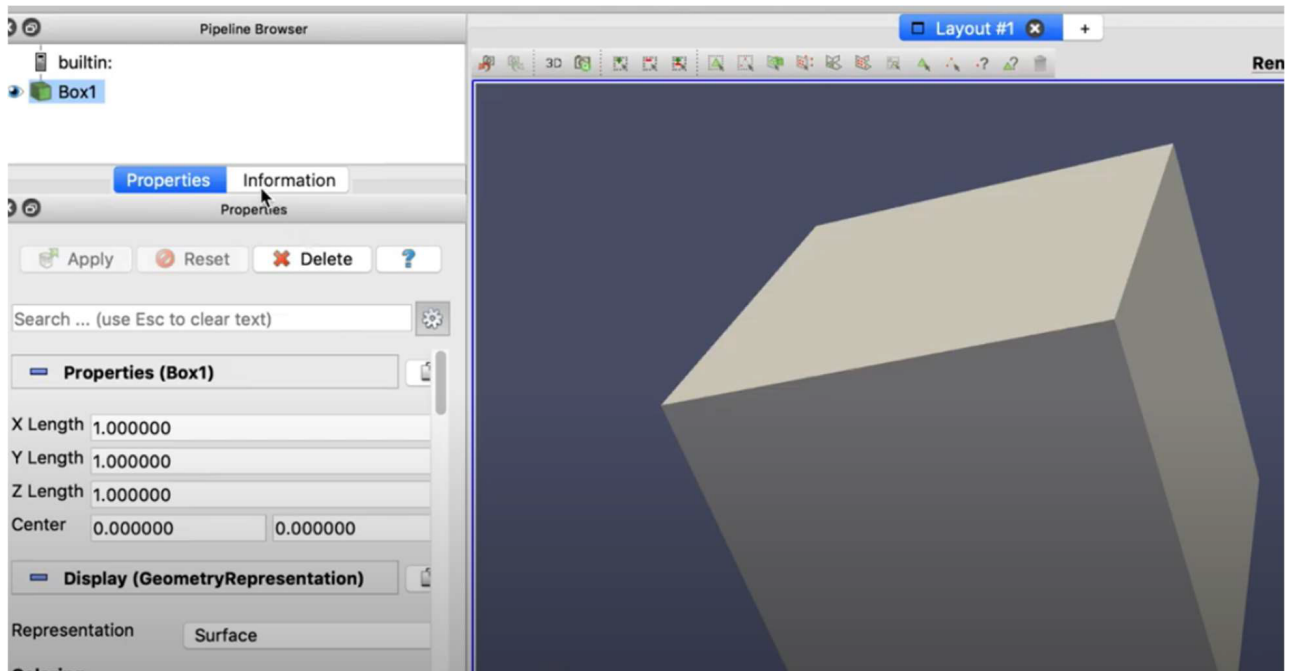


Рисунок 1.17 - Приклад рендерингу у VTK Reader

У клієнта є безліч налаштувань які зберігаються у $\$HOME/.config/ParaView$, що дозволяє легко переносити свої налаштування на інші клієнти.

У основних налаштуваннях можна налаштувати поведінку програми при закритті або при втраті зв'язку з сервером. Також можна зробити щоб усі зміни властивостей моделі відбувалися одразу без необхідності натискати на кнопку «apply». Усі налаштування можна зберігати у профіль та швидко перемикає між профілями. Таким чином можна мати різні профілі для роботи з різними типами моделей.

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

1.4.2. Програма інтерактивної наукової візуалізації ParaView

ParaView - це багатоплатформова програма з відкритим кодом для інтерактивної наукової візуалізації. Він має архітектуру клієнт-сервер для полегшення віддаленої візуалізації наборів даних та генерує моделі рівня деталізації (LOD) для підтримки інтерактивних частот кадрів для великих наборів даних. Це програма, побудована поверх бібліотек Visualization Toolkit (VTK).

ParaView - це програма, призначена для паралелізму даних на мультикомп'ютерах і кластерах із загальною та розподіленою пам'яттю. Його також можна запустити як однокomp'ютерну програму. ParaView розроблений як багат шарова архітектура, що базується на інструментарії візуалізації VTK.

Він забезпечує основу ParaView: представлення даних, алгоритми та механізм для з'єднання цих уявлень та алгоритмів разом для формування робочої програми.

Другий шар - це паралельні розширення набору інструментів візуалізації. Шар розпаралелювання VTK, розширений VTK для підтримки потокової передачі всіх типів даних та паралельне виконання на спільних та розподілених серверах.

Третій шар - це сам ParaView. ParaView надає графічний користувацький інтерфейс і прозора підтримує візуалізацію та рендеринг великих наборів даних через апаратне прискорення, паралелізм та прийоми деталізації. Кожен шар відповідає підмножині вимог і додає додаткову функціональність шару нижче.

ParaView надає графічний користувацький інтерфейс для інтерактивного дослідження великих наборів даних. Він створює таку функціональність на паралельному та розподіленому VTK. Оглянемо інструмент з перспективи користувача.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		27

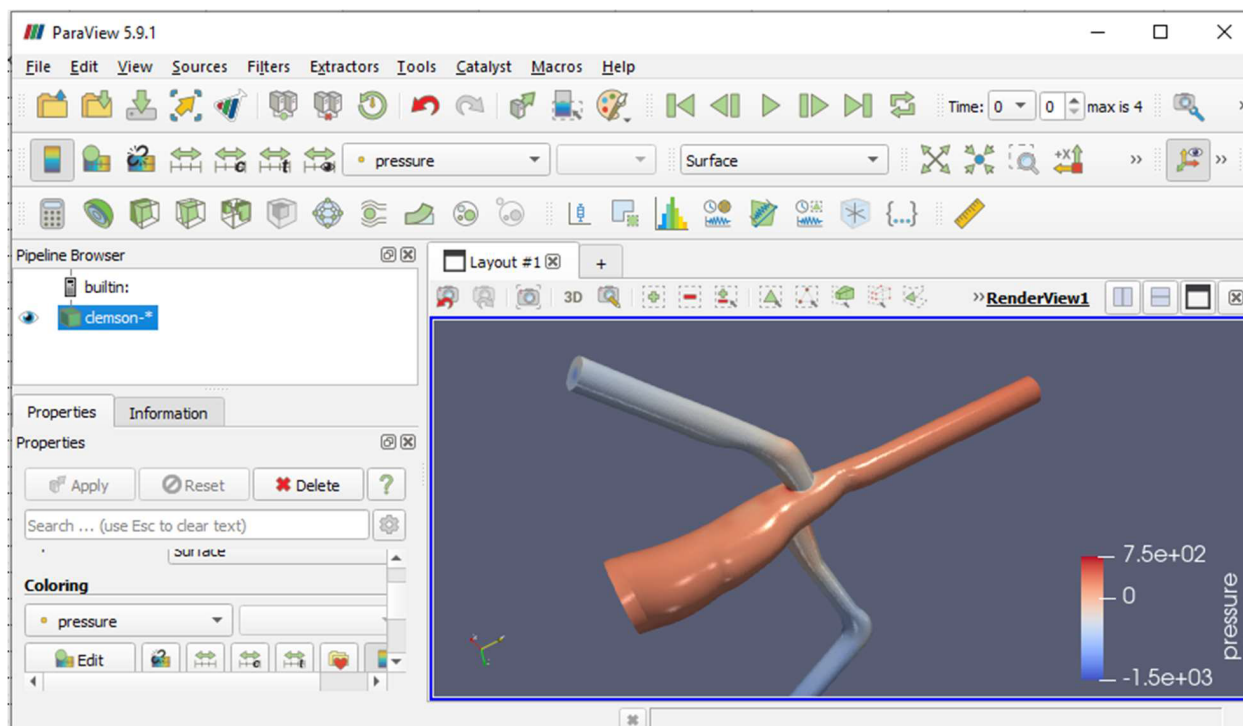


Рисунок 1.18 - Інтерфейс Paraview

Інтерфейс користувача має кілька панелей, що включають панель меню зверху програми, панель інструментів трохи нижче панелі меню, панель ліворуч, а область відображення справа. Кожна з цих областей описана докладніше нижче:

- Меню (зверху) - У верхній панелі меню є кнопки меню для завантаження та збереження даних, створення джерел та фільтрів, перегляд інших вікон, відображення довідки та інших стандартних можливостей;
- Панель інструментів - містить кнопки для роботи з камерою, перемикач між 2D і 3D режимами взаємодії, а також зміна центру обертання. Крім того, панель інструментів містить ярлики для створення екземплярів деяких часто використовуваних фільтрів.
- Ліва панель - Верхня частина цієї панелі містить вікно вибору або навігації. Вікно вибору містить список екземплярів джерел та фільтрів. Вікно навігації надає графічне представлення дій користувача над потоком даних. Під вікном вибору / навігації розташовані властивості джерел та фільтрів, а

також список властивостей. Властивості містять параметри модулів, такі як поточні значення ізоповерхні, обчислені модулем ізоповерхні.

- Область відображення - це місце, де відображається 3D-зображення сцени. У цій області передбачена взаємодія з мишею та клавіатурою.

ParaView має трирівневу клієнт-серверну архітектуру. Три логічні одиниці ParaView такі:

1. Клієнт - відповідає за встановлення візуалізації. клієнт контролює створення, виконання та знищення на серверах, але не отримує файлів даних (таким чином дозволяючи серверам масштабуватися не обмежуючись параметрами клієнта). У випадку використання графічного інтерфейсу, він також у клієнта. Клієнт завжди є послідовним додатком.

2. Сервер рендерингу - відповідає за візуалізацію. Сервер рендерингу може також бути паралельним

3. Сервер даних - відповідає за зчитування, фільтрацію та запис даних. Усі об'єкти, які бачимо в браузері, містяться на сервері даних.

Ці логічні одиниці не обов'язково повинні бути фізично розділені. Логічні одиниці часто вбудовані в одному додатку, усуваючи необхідність створення засобів будь-якого спілкування між ними.

1.4.3. Аналіз Eevee render

Blender Eevee - це потужна система візуалізації, розроблена для використання в програмі Blender. З його допомогою користувачі можуть створювати реалістичну 3D-графіку та анімацію. Однією з ключових особливостей Blender Eevee є альфа-рендерер, який дозволяє створювати зображення з прозорим фоном.

Альфа-рендерер – це функція яка дозволяє створювати сцени або об'єкти на прозорому фоні (рис. 1.19), що дуже корисно при роботі з логотипами, банерами, та різними анімаціями.

					ДРБ.ІІ – 29.00.00.000 ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		29

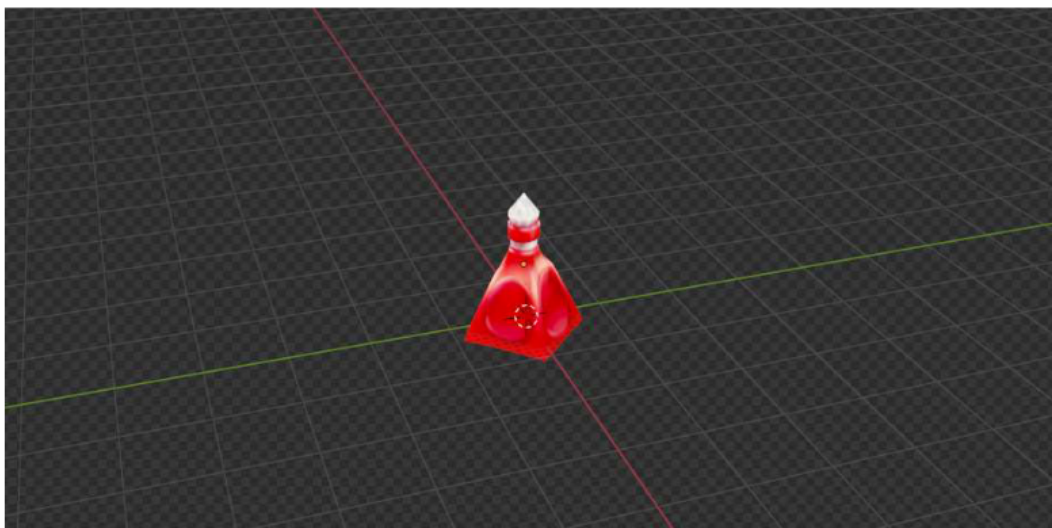


Рисунок 1.19 - Приклад об'єкту Eevee render на прозорому фоні

Також до особливостей цієї системи можна зазначити як рендер у реальному часі, мається на увазі те що при ввімкненні функції пре-рендеру, ми без будь яких очікувань отримуємо фінальний результат, тоді як наприклад для Cycles та Corona render ми маємо почекати доки розрахунки шляхів світла будуть оброблені.



Рисунок 1.20 – Приклад роботи рушія Eevee

Змн.	Арк.	№ докум.	Підпис	Дата

ДРБ.ІІ – 29.00.00.000 ПЗ

Арк.

30

Матеріали в Eevee створюються за допомогою тих самих вузлів шейдерів, що й Cycles, що полегшує рендеринг існуючих сцен. Для користувачів Cycles це робить Eevee чудовим рендерером у реальному часі як програмою попереднього перегляду. Шейдер Principled BSDF сумісний з багатьма ігровими движками, дозволяючи вам налаштовувати матеріали ігрових об'єктів безпосередньо в Blender (рис. 1.20).

1.4.4. Аналіз Cycles render

Cycles — це рушій рендерингу, який вперше з'явився у Blender версії 2.61 у грудні 2011. Він доступний у якості доповнень для програмного наряду з новим вбудованим рендером EEVEE. Cycles створює зображення методом трасування променів з упором на інтерактивність і простоту використання. Для матеріалів використовується система нодів, за допомогою якої можна створювати фотореалістичні матеріали будь-якої складності (рис. 1.21).

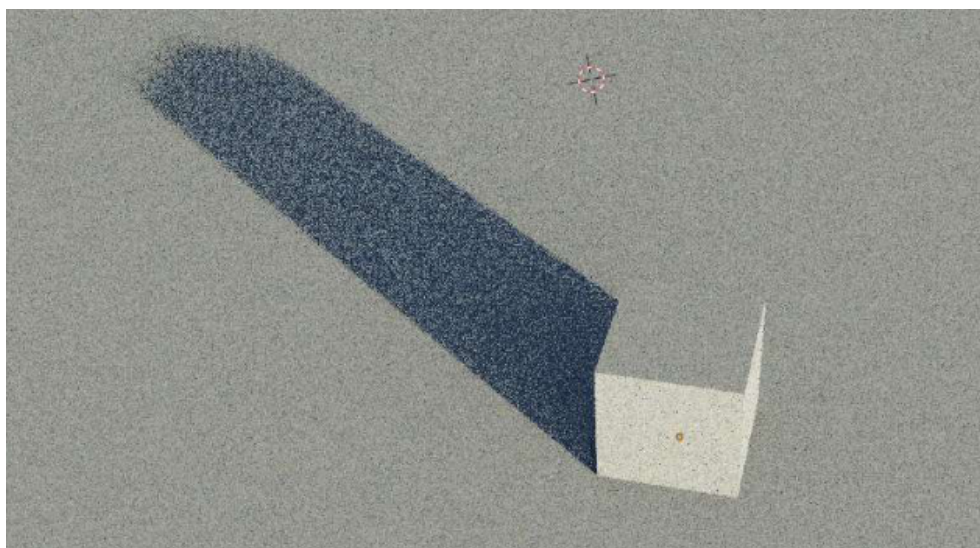


Рисунок 1.21 – Приклад пре-рендеру у Cycles

Найбільшими перевагами рушія є можливість швидкого перегляду результату безпосередньо у вікні 3D-вида, а також рендеринг за допомогою графічної карти (GPU).

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		31

Рушій рендеринга Cycles має в своєму розпорядженні більше восьмидесяти нодів для створення матеріалів. Cycles використовує систему під назвою «Інтегратор» це алгоритм візуалізації, який використовується для розрахунку освітлення. Наразі Cycles підтримує інтегратор із трасуванням шляху та прямим вибіркоким світлом. Він добре працює для різних налаштувань освітлення, але не дуже підходить для каустики та іншого складного освітлення.

1.4.4. Аналіз Corona render

Corona Renderer — це фото-реалістичний візуалізатор, розроблений у 2009 році. На даний момент один із найпопулярніших візуалізаторів для 3Ds max. Завдяки своїм перевагам, Corona render, є одним з найкращих систем для візуалізації.

Головною перевагою цього рушія є якісна та реалістична візуалізація. Одним з найголовніших факторів серед них є робота з освітленням, при роботі з будь-яким елементом середовища Corona Renderer дозволяє системі Sun and Sky регулювати перспективу, створювати ефект об'єму. Можливе встановлення параметрів погоди і висоту сонця над горизонтом.

Функція Interactive LightMix дозволяє змінювати інтенсивність світла та колір як під час, так і після візуалізації. Візуалізація різного часу доби за допомогою одного або кількох джерел світла доступна за змін лише декількох параметрів. Налаштування можна зберегти для повторного використання для відтворення сцени в тому самому місці, але при іншому освітленні.

Відбиття світлових променів, викликане вигнутими поверхнями, допомагає досягти більшої реалістичності візуалізації. Яскравий приклад каустики - гра світла і тіні на дні моря або басейну. Corona Renderer дозволяє оптимізувати роботу з додавання ефекту. Надбудова Fast Caustics Solver автоматично обчислює каустики для всіх матеріалів у сцені на даний момент.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

Базові матеріали які входять цього рушія рендера, вже мають в собі налаштування під різні поверхні, з своїми властивостями, таких матеріалів нараховується близько тридцяти п'яти одиниць, до них входять, скло, шкіра, метали, дерев'яні поверхні.



Рисунок 1.22 - Приклад візуалізації Corona render

За усіма параметрами на даний час Corona render самий універсальний рушій рендеру, що дозволяє як дуже просто налаштовувати параметри так і змінювати фінальний результат вже після завершення роботи.

1.4.5. Веб-платформа комп'ютерної графіки Shadertoy

Shadertoy.com — веб-платформа для ентузіастів комп'ютерної графіки, що призначена для експериментів із технологіями візуалізації та процедурним мистецтвом за допомогою шейдерів на GLSL, та інтернетспільнота, що утворилася навколо проекту. Основна технологія проекту — WebGL (Web Graphics Library) — графічне API, що базується на OpenGL ES, та призначене для доступу до апаратноприскореного 3Дрендерингу у браузері без використання сторонніх плагінів. WebGL підтримується всіма основними веб-браузерами, тому проведення

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		33

експериментів не потребує розгортання середовища розробки чи іншої підготовки.

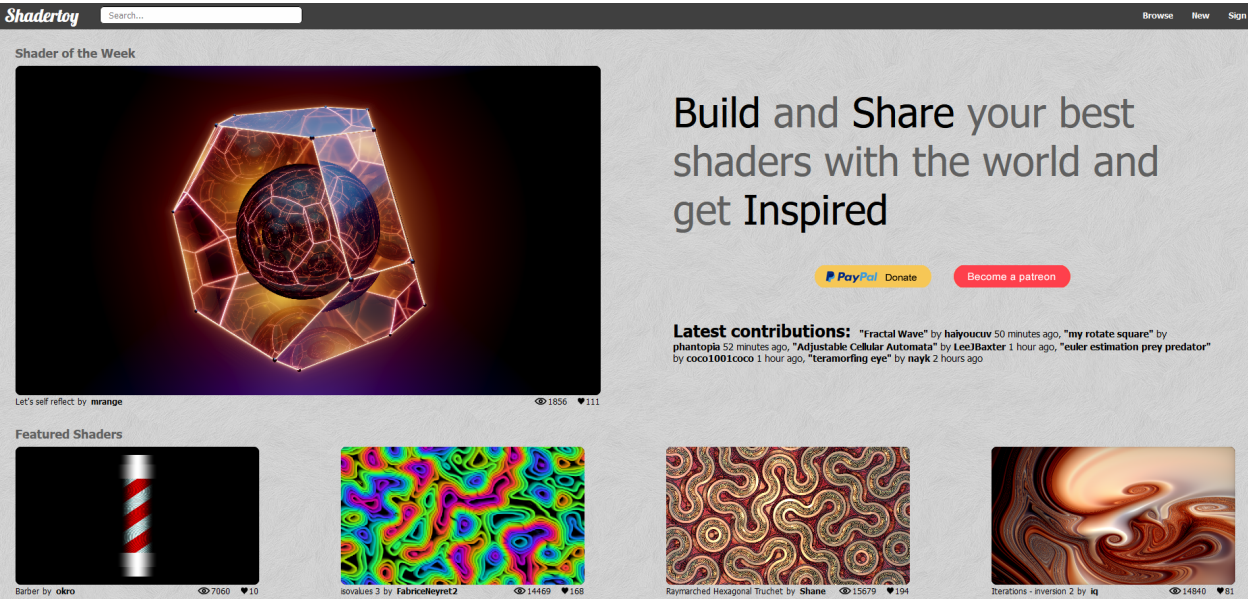


Рисунок 1.23 - Веб-платформа Shadertoy



Рисунок 1.24 — Процедурно згенерована за допомогою знакових полів відстані (signed distance fields) анімація

Shadertoy є інструментом, що дозволяє найбільш швидкі ітерації, що особливо важливо для експериментів з процедурної генерації — власне, єдина функція, що надається даним інструментом — це редагування піксельних шейдерів з можливістю по бачити результат їх роботи у режимі реального часу.

Проте настільки велика швидкість ітерацій неминуче тягне за собою ряд компромісів та обмежень, які унеможливають використання цього інструменту для скільки не будь масштабних проектів. Серед основних обмежень функціональності можна назвати:

- можливість редагувати лише піксельні шейдери, тобто немає засобів контролю над графічним конвейєром (graphics pipeline);
- обмеженість WebGL як графічного API, відсутність підтримки сучасних технологій рендерингу, наприклад, апаратноприскореного рейтрейсингу;
- обмеженість у використанні довільних асетів, наприклад, немає можливості використовувати завантажені користувачем текстури.

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		35

2. МОДЕЛЮВАННЯ ТА РОЗРОБКА СТРУКТУРИ СИСТЕМИ

ГРАФІЧНОГО МОДЕЛЮВАННЯ

2.1. Аналіз вимог до розроблюваного програмного забезпечення

Головним функціоналом розроблюваного проекту є саме рендеринг об'ємних моделей. Перш ніж почати рендерити свої моделі, користувачу необхідно зареєструватися у системі за допомогою логіну та паролю, або авторизуватися, якщо він уже має акаунт. При створенні замовлення на рендеринг, користувачу необхідно завантажити свою модель та обрати певні налаштування для сцени.

Діаграми варіантів використання (Use Case Diagrams) є інструментом UML (Unified Modeling Language) і широко використовуються при аналізі та проектуванні програмного забезпечення. Діаграми варіантів використання є особливо корисними на ранніх етапах розробки ПЗ, допомагаючи забезпечити, що система задовольняє всі вимоги користувачів і бізнесу.

Діаграма варіантів використання для двох типів користувачів представлена на рисунках 2.1 і 2.2.

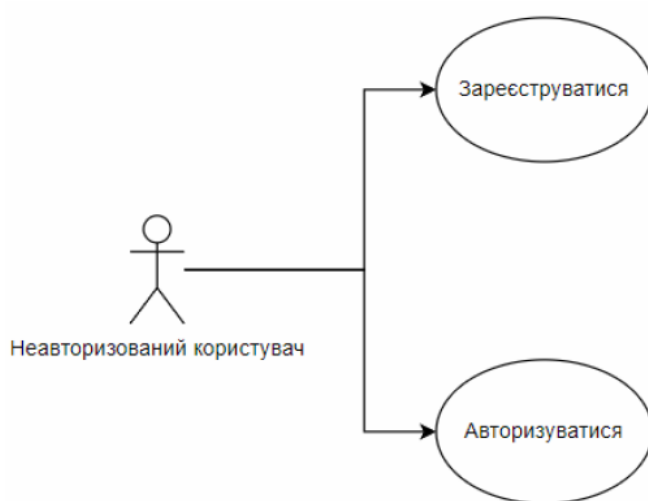


Рисунок 2.1 - Діаграма варіантів використання для неавторизованого користувача

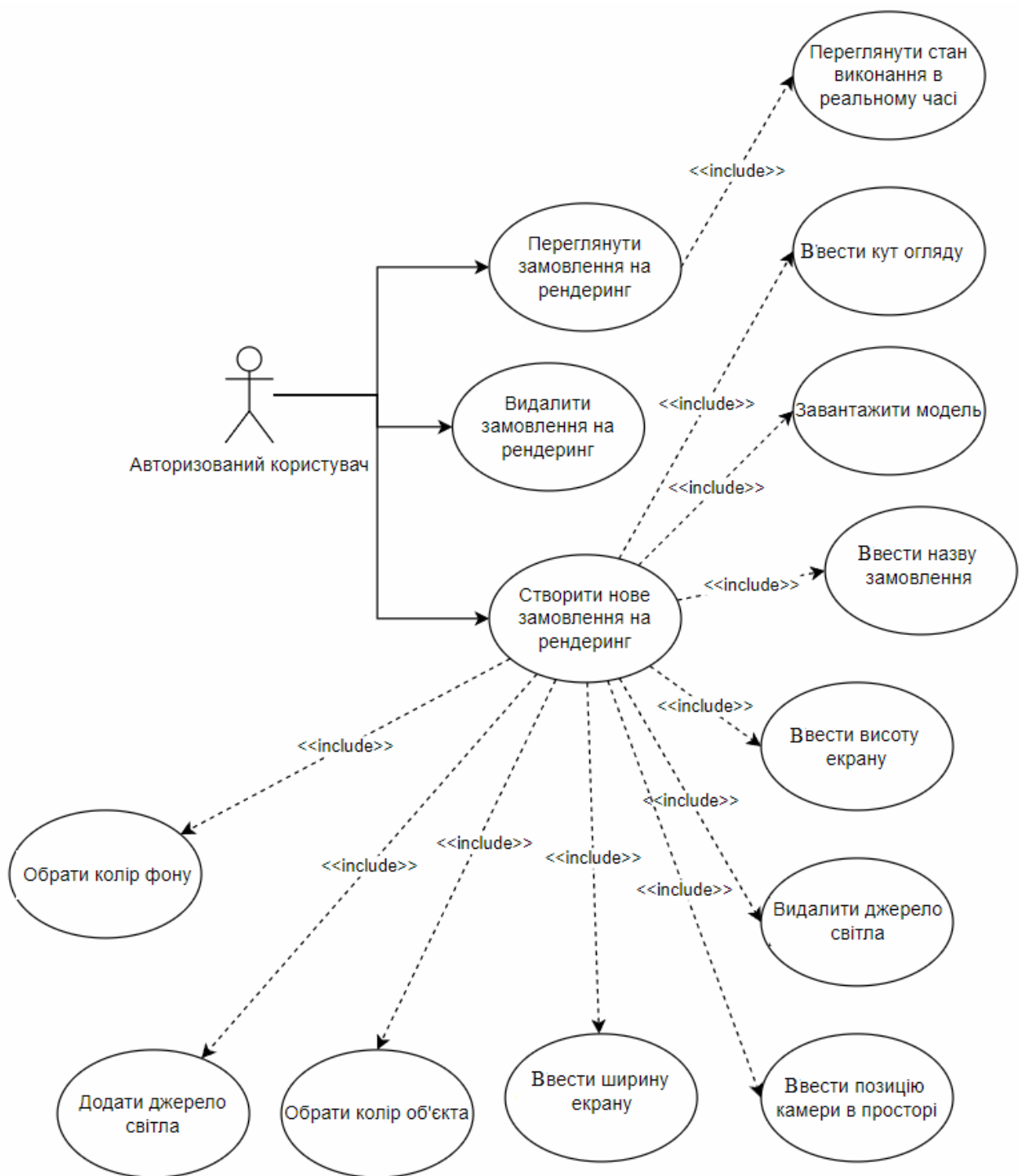


Рисунок 2.2 - Діаграма варіантів використання для авторизованого користувача

У таблицях 2.1 – 2.5 наведено варіанти використання проекту розробки програмного забезпечення системи графічного моделювання.

Таблиця 2.1 – Варіант використання реєстрації нового користувача

Назва	Реєстрація нового користувача
Основні дії користувача	Учасник надсилає HTTP запит з методом POST з тілом запиту у вигляді електронної пошти та пароля у форматі JSON. У відповіді отримує пару Access Token та Refresh Token, та профіль користувача.
Очікуваний результат	- Успішна реєстрація користувача - У випадку непроходження валідації введених даних, користувачу повертається

Таблиця 2.2 – Варіант використання авторизації користувача

Назва	Авторизація користувача
Основні дії користувача	Учасник надсилає HTTP запит з методом GET з тілом запиту у вигляді електронної пошти та пароля у форматі JSON. У відповіді отримує пару Access Token та Refresh Token, та профіль користувача.
Очікуваний результат	- Успішна авторизація - У випадку непроходження валідації введених даних, користувачу повертається

Таблиця 2.3 – Варіант використання створення нового замовлення на рендеринг

Назва	Створення нового замовлення на рендеринг
Основні дії користувача	Учасник надсилає HTTP запит з методом POST, де містяться заголовки «Authorization» зі значенням Access Token, та «Content-Type» зі значенням «multipart/form-data». Запит повинен містити форму, де у полі Model

	повинен знаходитись файл моделі, а у полі Order – замовлення у форматі JSON. Замовлення на рендеринг повинно містити назву замовлення, налаштування екрану, кольору об'єкта та заднього фону, камери, та список джерел світла.
Очікуваний результат	Створення замовлення на рендеринг

Таблиця 2.4 – Варіант використання отримання списку замовлень на рендеринг

Назва	Отримання списку замовлень учасника на рендеринг
Основні дії користувача	Учасник надсилає HTTP запит з методом GET, де міститься заголовок «Authorization» зі значенням у вигляді Access Token. У відповіді отримує список замовлень користувача, якому належить Access Token. Якщо вказаний Access Token є невалідним, то відповідь матиме код стану 401 Unauthorized.
Очікуваний результат	У відповіді міститься список отриманих замовлень користувача

Таблиця 2.5 – Варіант використання перегляду стану замовлення

Назва	Перегляд стану замовлення в реальному часі
Основні дії користувача	Учасник надсилає HTTP запит з методом GET, де міститься заголовок «Authorization» зі значенням у вигляді Access Token. У відповіді отримує список замовлень користувача, якому належить Access Token.
Очікуваний результат	Учасник отримує повідомлення при зміні стану певного замовлення

Таблиця 2.6 – Варіант використання видалення замовлення на рендеринг

Назва	Видалення замовлення на рендеринг
Основні дії користувача	Учасник надсилає HTTP запит з методом DELETE, де міститься заголовок «Authorization» зі значенням у вигляді Access Token. Кінцевий сегмент URL повинен містити ідентифікатор замовлення, яке необхідно видалити. Відповідь має статус код 204 No Content.
Очікуваний результат	Видалення вказаного замовлення із бази даних

2.2. Розробка бізнес моделі

На рисунку 2.3 зображено BPMN діаграму, що описує бізнес процес рендерингу зображення системи графічного моделювання від отримання нового замовлення до отримання готового результату.

Опис послідовності створення замовлення та рендерингу об'ємної моделі:

- клієнт надсилає на сервіс замовлень файл моделі та налаштування замовлення;

- якщо хоча б одне поле у налаштуваннях не пройшло валідацію, то сервіс замовлень повертає клієнту список помилок;

- якщо поля пройшли валідацію, то сервіс замовлень завантажує файл моделі у хмарне сховище до бакету з моделями. У результаті сервіс замовлень отримує посилання на файл у сховищі;

посилання на модель разом із налаштуваннями зберігаються у базу даних;

- сервіс замовлень створює повідомлення про нове замовлення та кладе його у чергу у брокер повідомлень;

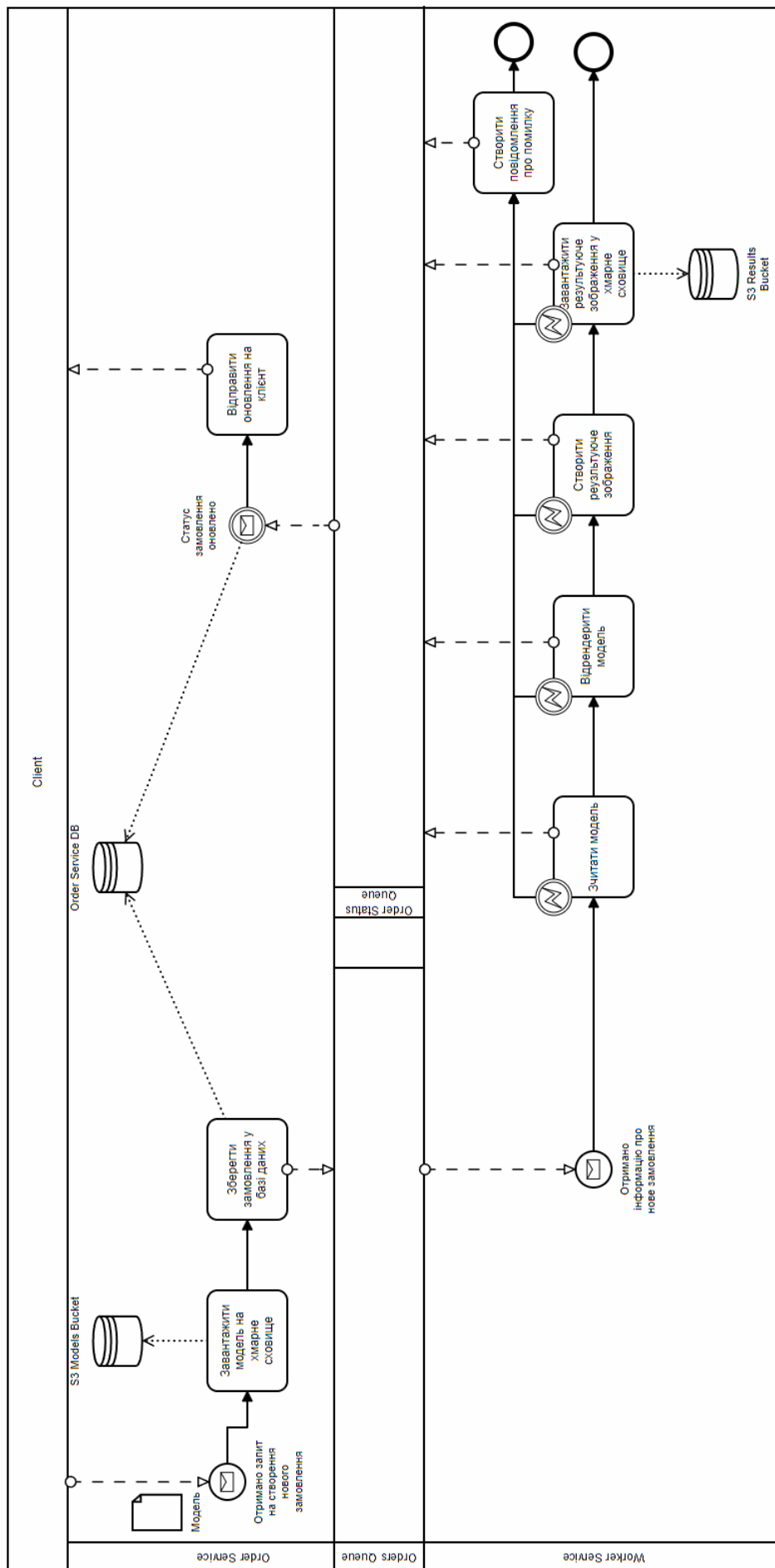


Рисунок 2.3 - Опис бізнес процесу рендерингу зображення

Змн.	Арк.	№ докум.	Підпис	Дата

ДРБ.ІІ – 29.00.00.000 ПЗ

– один із вільних воркерів, що підписаний на оновлення даної черги, отримує повідомлення;

воркер отримує із повідомлення налаштування замовлення та завантажує із хмарного сховища файл моделі;

– воркер створює відповідне повідомлення та надсилає його у брокер повідомлень.

Варто зазначити, що будь-яке повідомлення, що надсилається воркером, являє собою модель замовлення та обробляється сервісом замовлень, який оновлює запис відповідного замовлення у базі даних. Після оновлення, сервіс замовлень пересилає повідомлення на сторону клієнта за допомогою SSE. Таким чином забезпечується оновлення стану замовлення в реальному часі;

– воркер намагається зчитати модель та отримати список полігонів;

– якщо воркер не може зчитати файл, або він не містить полігонів, то створює повідомлення про відповідну помилку та кладе його у чергу у брокер повідомлень;

– якщо зчитування успішне, то воркер присвоює замовленню відповідний статус та кладе повідомлення у брокер повідомлень;

– воркер починає рендеринг отриманих полігонів з файлу моделі з вказаними у замовленні налаштуваннями;

– результатом рендерингу є бітмапа, де кожному пікселю екрану присвоєний певний колір. Воркер створює відповідне повідомлення для брокера повідомлень;

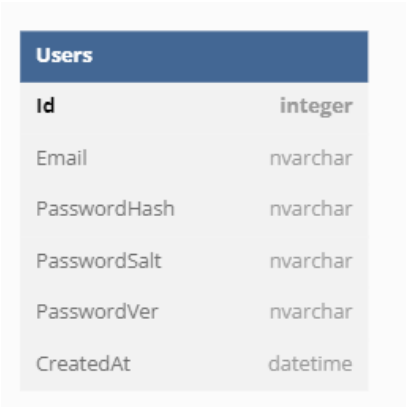
– наступним кроком є перетворення отриманої бітмапи у результуюче зображення формату .jpeg;

– воркер завантажує отримане зображення до хмарного сховища S3 у бакет з результатами рендерингу і створює останнє повідомлення для брокера повідомлень.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

2.3. Розробка та проектування структури бази даних

Сервіс авторизації використовує дві бази даних. Для зберігання користувачів використовується база даних Users, що знаходиться у СУБД SQL Server. На рисунку 2.4 показана структура, а в таблиці 2.7 наведено опис таблиці Users.



Users	
Id	integer
Email	nvarchar
PasswordHash	nvarchar
PasswordSalt	nvarchar
PasswordVer	nvarchar
CreatedAt	datetime

Рисунок 2.4 – Таблиця Users

Таблиця 2.7 – Опис таблиці Users

Назва стовпця	Тип даних	Опис
Email	Nvarchar(max)	Електронна пошта користувача
PasswordHash	Nvarchar(max)	Хеш паролю користувача
PasswordSalt	Nvarchar(max)	Сіль до хешу паролю
PasswordVer	Nvarchar(max)	Версія алгоритму хешування
CreatedAt	Datetime2(7)	Дата створення облікового запису

Для зберігання Refresh Token користувачів використовується база даних у СУБД Redis. Таблиці у Redis представлені у вигляді ключ-значення.

В якості ключа було взято Refresh Token у вигляді рядку. У якості значення використано модель RefreshTokenData, яку було серіалізовано у файл формату .JSON. На рисунку 2.5 показано клас RefreshTokenData.

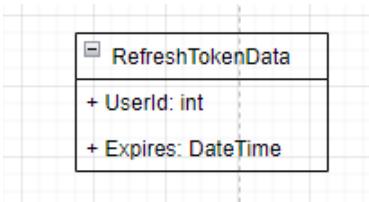


Рисунок 2.5 – Клас RefreshTokenData

Для зберігання даних про замовлення, сервіс використовує базу даних SQL Server, модель розробленої структури БД подано на рисунку 2.6.



Рисунок 2.6 – Модель бази даних для замовлень

У таблицях 2.8 – 2.13 наведено опис таблиць, що входять до бази даних.

Таблиця 2.8 – Опис таблиці LightSettings

Назва стовпця	Тип даних	Опис
OrderId	int	Ідентифікатор замовлення
PositionX	Real	Положення джерела світла по осі абсцис
PositionY	real	Положення джерела світла по осі ординат
PositionZ	Real	Положення джерела світла по осі аплікату
Intensity	Int	Інтенсивність світла

Таблиця 2.9 – Опис таблиці ScreenSettings

Назва стовпця	Тип даних	Опис
Id	Int	Ідентифікатор налаштування
OrderId	int	Ідентифікатор замовлення
Width	Int	Ширина екрану
Height	Int	Висота екрану
Fov	Int	Кут огляду

Таблиця 2.10 – Опис таблиці ObjectColorSettings

Назва стовпця	Тип даних	Опис
Id	Int	Ідентифікатор налаштування
OrderId	int	Ідентифікатор замовлення
Red	real	Рівень червоного
Green	Real	Рівень зеленого
Blue	real	Рівень синього

Таблиця 2.11 – Опис таблиці Orders

Назва стовпця	Тип даних	Опис
Id	Int	Ідентифікатор налаштування
UserId	Int	Ідентифікатор користувача, якому належить замовлення
Name	Nvarchar(max)	Ім'я замовлення
Status	Nvarchar(max)	Статус замовлення
Result	Nvarchar(max)	Посилання на файл результуючого зображення
FaultMessage	Nvarchar(max)	Опис помилки при замовленні
CurrentAction	Nvarchar(max)	Опис поточної дії над замовленням
CreatedAt	Datetime2(7)	Дата створення замовлення
Completed	Datetime2(7)	Дата завершення замовлення

Таблиця 2.12 – Опис таблиці CameraPositionSettings

Назва стовпця	Тип даних	Опис
Id	Int	Ідентифікатор налаштування
OrderId	int	Ідентифікатор замовлення
PositionX	Real	Положення камери по осі абсцис
PositionY	real	Положення камери по осі ординат
PositionZ	Real	Положення камери по осі аплікват

Таблиця 2.13 – Опис таблиці BackgroundColorSettings

Назва стовпця	Тип даних	Опис
Id	Int	Ідентифікатор налаштування
OrderId	int	Ідентифікатор замовлення
Red	real	Рівень червоного
Green	Real	Рівень зеленого
Blue	real	Рівень синього

2.4. Реалізація алгоритмів та функціоналу процесів реєстрації та авторизації

Процес реєстрації є важливим етапом взаємодії користувача з програмним забезпеченням, оскільки від його простоти та зручності часто залежить перше враження користувача про продукт. Нижче наведено основні особливості процесу реєстрації при розробці програмного забезпечення:

- Простота і зручність;
- Підтвердження електронної пошти;
- Захист від автоматизованих реєстрацій (CAPTCHA, Anti-Bot);
- Підтримка соціальних мереж (SSO);
- Мобільна верифікація;
- Парольні політики.

RegistrationService – клас-сервіс, що відповідає за реєстрацію користувачів в системі за введеними електронною поштою і паролем. Методи даного класу наведені в таблиці 2.14.

Таблиця 2.14 – Методи класу RegistrationService

Назва методу	RegisterUser
Опис параметрів	Електронна пошта та пароль у вигляді рядків
Опис методу	Реєструє нового користувача в системі
Результат	Пара Access Token і Refresh Token, а також профіль користувача
Назва методу	ValidateUserEmail
Опис параметрів	Електронна пошта у вигляді рядка
Опис методу	Перевіряє, чи існує користувач з введеною поштою у базі даних
Результат	True, якщо такий користувач існує, у протилежному випадку false

На рисунку 2.7 зображено алгоритм реєстрації.

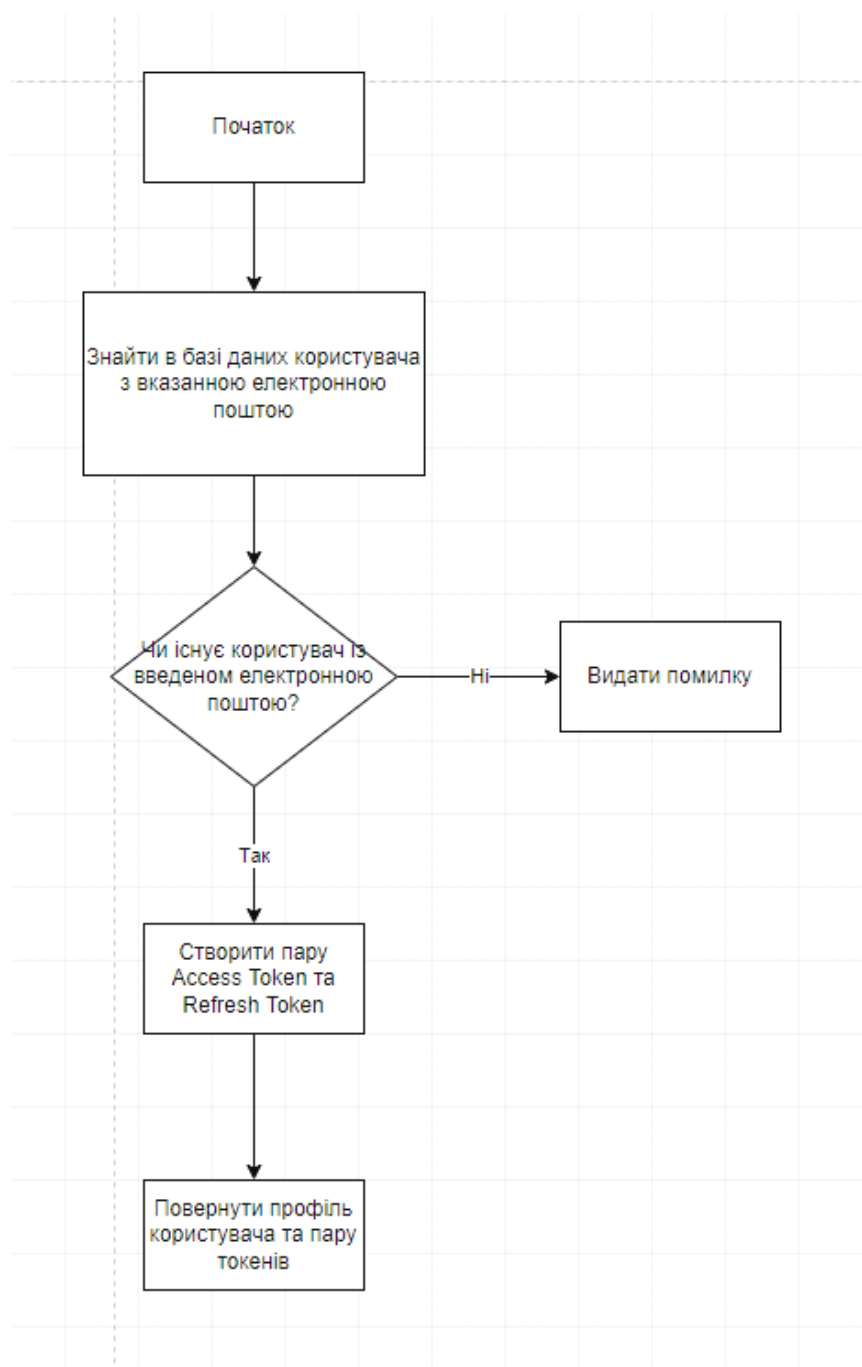


Рисунок 2.7 – Алгоритм реєстрації

Процес авторизації є критично важливим етапом розробки програмного забезпечення, оскільки він визначає, які ресурси та функції доступні конкретним користувачам. Авторизація відрізняється від автентифікації, яка

полягає в перевірці особи користувача. Основні особливості процесу авторизації включають:

- Ролі та дозволи;
- Контроль доступу на основі ролей (RBAC - Role-Based Access Control);
- Контроль доступу на основі атрибутів (ABAC - Attribute-Based Access Control - модель, що використовує атрибути користувачів (наприклад, відділ, місцезнаходження, час доби) для прийняття рішень щодо доступу);
- Гнучкість: Дозволяє більш детальне налаштування політик доступу, ніж RBAC;
- Делегована авторизація (OAuth, користувачі можуть контролювати, які дані доступні для сторонніх додатків);
- Контроль доступу на основі політик (PBAC - Policy-Based Access Control);
- Сумісність та інтеграція (сумісність з існуючими системами, API та SDK).

AuthorizationService – клас-сервіс, який авторизує користувача в системі за введеними електронною поштою та паролем. Методи даного класу наведено в таблиці 2.15.

Таблиця 2.15 – Методи класу AuthorizationService

Назва методу	Authorize
Опис параметрів	Електронна пошта та пароль у вигляді рядків
Опис методу	Авторизує користувача в системі
Результат	Пара Access Token та Refresh Token, а також профіль користувача

На рисунку 2.7 зображено алгоритм авторизації.

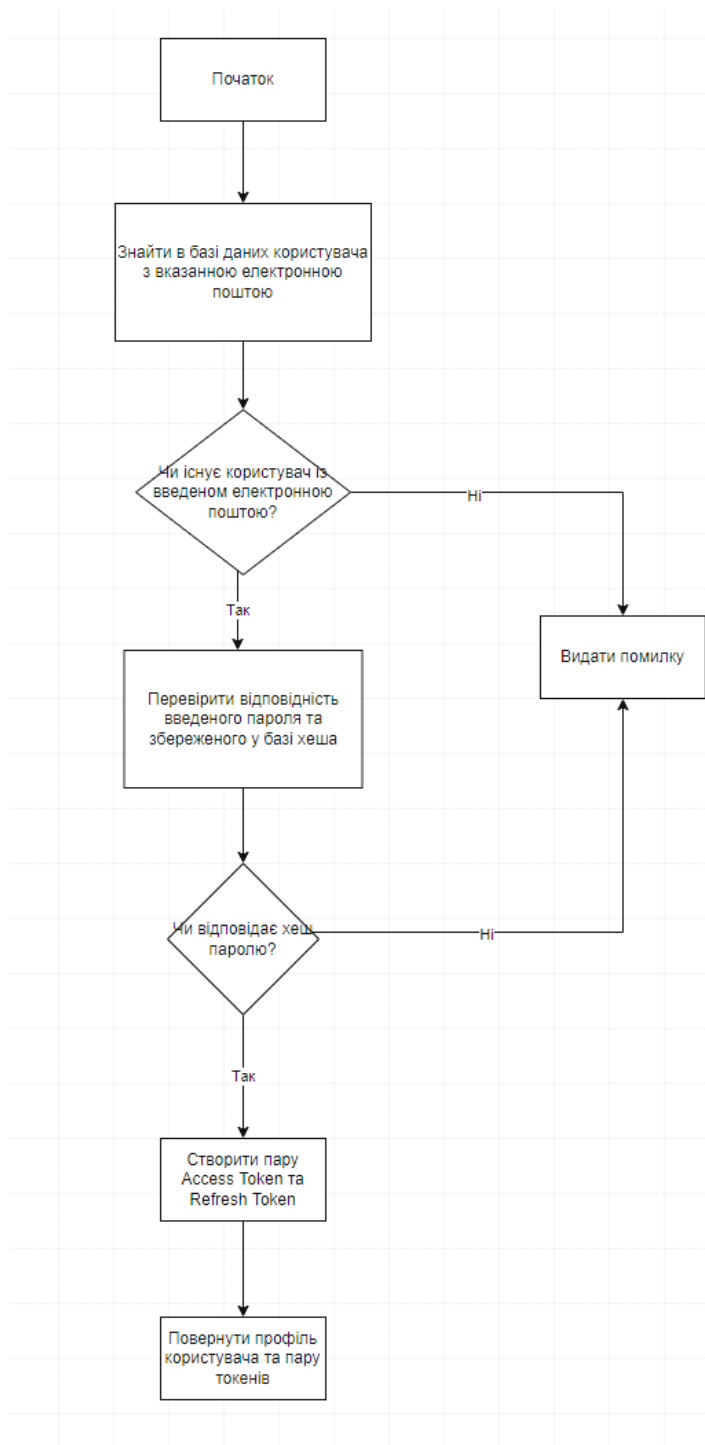


Рисунок 2.7 – Алгоритм авторизації

2.5. Вибір архітектурного рішення для системи графічного моделювання

Ріст будь якого веб-ресурсу означає збільшення кількості користувачів, а це призводить до збільшення кількості запитів, які цей веб-ресурс повинен

мати змогу опрацювати щосекунди. Для подолання даної проблеми, зазвичай, використовують масштабування.

Використання мікросервісної архітектури є одним з часто використовуваних методів горизонтального масштабування на даний момент. Мікросервісна архітектура має доволі значний недолік – збільшення значення затримок при комунікації між сервісами. Тобто, при ідентичній реалізації сервісів, монолітна архітектура може мати кращі показники швидкодії, ніж мікросервісна. З іншого боку, мікросервісна архітектура дає більше можливостей для масштабування, що є суттєвою перевагою для систем, що швидко розвиваються.

Таким чином, як мікросервісна, так і монолітна архітектури мають як позитивні, так і негативні властивості, тому визначення архітектури, яка буде використана при побудові веб-сервісів, дуже важливе і має безпосередній вплив на те, як система буде розвиватись надалі.

Аналіз існуючих джерел інформації показує, що перед використанням мікросервісної архітектури необхідно провести аналіз розроблюваної системи і визначити, наскільки прийнятними є показники затримки при комунікації між веб-сервісами, і чи не призведуть вони до низької швидкодії системи і неможливості подальшого її масштабування. Таким чином, важливо оцінити обґрунтованість використання мікросервісної архітектури для побудови сервісів конкретного веб-ресурсу.

Монолітна архітектура (рис. 2.8) є традиційним підходом до розробки програмного забезпечення, де всі компоненти додатку (інтерфейс користувача, бізнес-логіка, база даних тощо) інтегровані в один великий, нероздільний блок. Цей підхід включає створення, розгортання та масштабування всієї програми як єдиного цілого. Через проблеми при побудуванні такої архітектури, багато організацій почали шукати альтернативні підходи, які б могли забезпечити більшу гнучкість, масштабованість та швидкість розгортання.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

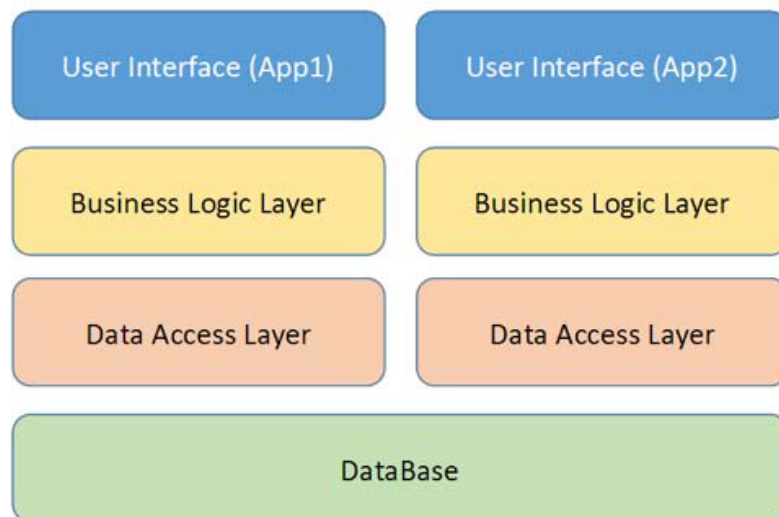


Рисунок 2.8 - Програмна реалізація монолітної архітектури

Мікросервісна архітектура використовує інший підхід. Кожен сервіс реалізований як окрема програмна система, часто у кожного сервісу є своя база даних. Оскільки сервіси не мають доступу до бази даних іншого сервісу – доступ до таких даних здійснюється викликом інших сервісів ресурсу. Часто сервіси групують, якщо вони реалізують схожий, або тісно пов'язаний функціонал. Прикладом такої ситуації може бути процес реєстрації і аутентифікації. Так, за ці два процеси будуть відповідати кілька сервісів, але оскільки вони будуть працювати з сутністю «Користувач», має сенс використовувати загальну базу даних. Приклад реалізації мікросервісного підходу надано на рисунку 2.9.

Залежно від значень характеристик, описаних вище, можна визначити яка архітектура найкраще підходить до кожного конкретного проекту.

Для оцінки буде використана система рейтингу. В таблиці 2.16 відображений рейтинг значень характеристик. Опис значень, які можуть отримати монолітна та мікросервісна архітектури в залежності від значень характеристик:

- -1 – вибір архітектури негативно вплине на розвиток проекту;
- 0 – вибір архітектури не буде впливати на подальшу розробку проекту;

- +1 – вибір архітектури позитивно вплине на розвиток проекту;
- +2 – вибір архітектури значним чином позитивно вплине на розвиток проекту.

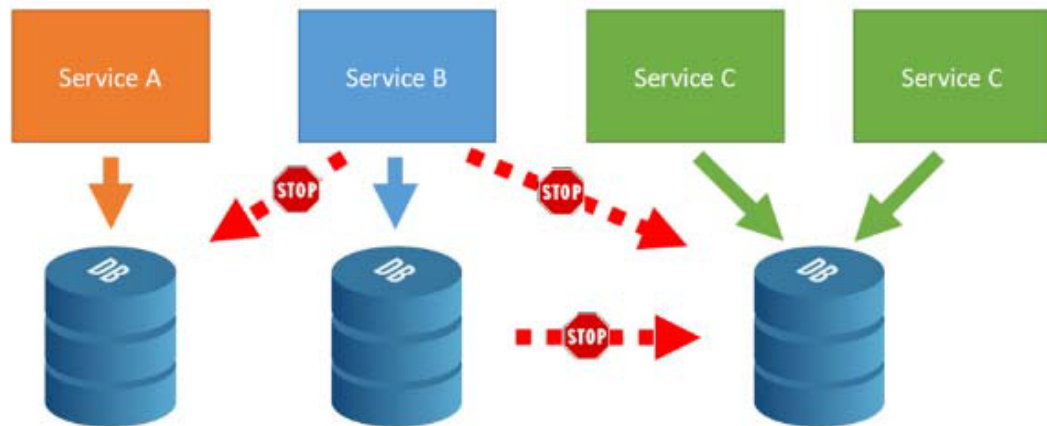


Рисунок 2.9 - Програмна реалізація мікросервісної архітектури

Таблиця 2.16 - Рейтинг значень характеристик проекту та його впливу на вибір архітектури

№	Назва характеристики	Значення характеристики	Монолітна архітектура	Мікросервісна архітектура
1	Вимоги до швидкодії	Відсутні	+1	0
		Не критично	0	0
		Критично	0	+2
2	Розмір команди	Маленька	+1	-1
		Середня	0	0
		Велика	-1	+2
3	Частота встановлення нових версій	Відразу після розробки	-1	+1
		По графіку	0	0
4	Технологічна орієнтованість команди	Вузька	+1	0
		Різностороння	-1	+1
5	Досвід роботи з мікросервісами	Є	0	+1
		Немає	0	-1

Значення рейтингу обумовлені наступними факторами:

1) вимоги до швидкодії – при відсутності вимог до швидкодії, висока складність розробки мікросервісів не обґрунтована. З іншого боку, оскільки у мікросервісів кращі показники масштабування, можна отримати суттєвий виграш при їх використанні, де швидкодія дійсно важлива;

2) при використанні мікросервісів з маленькою командою втрачається виграш в розділенні відповідальності між сервісами, а при використанні монолітів з великою командою – кожен член команди має дуже мало інформації про систему, за яку він відповідальний;

3) мікросервіси показують себе краще при частому встановленні нової версії, оскільки воно здійснюється швидко і не блокує решту сервісів;

4) при різносторонній технологічній орієнтованості команди, мікросервіси є кращими, оскільки кожен член команди має можливість використовувати ті технології, в яких у нього найбільше знань та досвіду;

5) очевидно, що відсутність досвіду в використанні мікросервісів негативно вплине на процес розроблення. Підсумовуючи результати аналізу (табл. 2.16), слід визначити, що мікросервісна архітектура має суттєво вищий рейтинг, якщо скласти значення за усіма показниками. Цей результат може бути уточнений, якщо використати вагові коефіцієнти для окремих показників.

У контексті порівняльного аналізу між мікросервісною та монолітною архітектурами, застосовуються численні методи та підходи для глибокого аналізу. До квалітативних методів належать експертні оцінки від досвідчених розробників або архітекторів, які розглядають переваги та недоліки кожної архітектури, а також кейс-стаді, які забезпечують детальний аналіз реальних проектів. У кількісному аналізі використовуються метрики продуктивності, такі як час відгуку та пропускна здатність, а також аналіз вартості та ефективності, що включає витрати на розробку, підтримку та масштабування.

Методи моделювання та симуляції включають архітектурне моделювання для створення моделей кожної архітектури та проведення симуляцій навантаження, щоб оцінити масштабованість і стійкість. Технічний аналіз охоплює аналіз залежностей, складності, міжсервісної комунікації та безпекових ризиків. Окремий фокус складає аналіз вимог та

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

потреб користувачів, який включає оцінку задоволеності користувачів та вимог до функціональності, аналізуючи, наскільки кожна архітектура відповідає бізнес-вимогам і потребам користувачів.

При проведенні порівняльного аналізу між мікросервісною та монолітною архітектурами, важливо враховувати різноманітні критерії оцінки. До основних показників продуктивності належать час відгуку системи, пропускна здатність при різних навантаженнях та ефективність використання ресурсів, як-от CPU та пам'ять.

Масштабованість оцінюється через легкість додавання нових компонентів або сервісів та здатність системи підтримувати зростаючий обсяг даних та користувачів. Гнучкість та адаптивність визначаються спроможністю архітектури адаптуватися до змін у бізнес-процесах та легкістю внесення змін і додавання нових функцій.

Сумісність та інтеграція охоплюють легкість інтеграції з іншими системами, підтримку стандартів та протоколів. Витрати на розробку та підтримку включають початкові та тривалі витрати, а також витрати на оновлення та масштабування.

Заключним елементом є порівняння життєвого циклу розробки, де аналізується швидкість розробки та гнучкість змін і масштабування обох архітектур. Цілісне застосування цих різноманітних методів дає змогу отримати всебічне та об'єктивне розуміння переваг та недоліків кожної архітектури.

Зважаючи на всі переваги було вирішено використати мікросервісний підхід для вирішення поставленої задачі.

Кожен з мікросервісів був побудований з використанням трьохшарової архітектури. Шар являє собою абстракцію, яка слугує для розмежування методів за їх призначенням. Кожен шар має доступ лише до методів сусіднього шару нижнього рівня.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

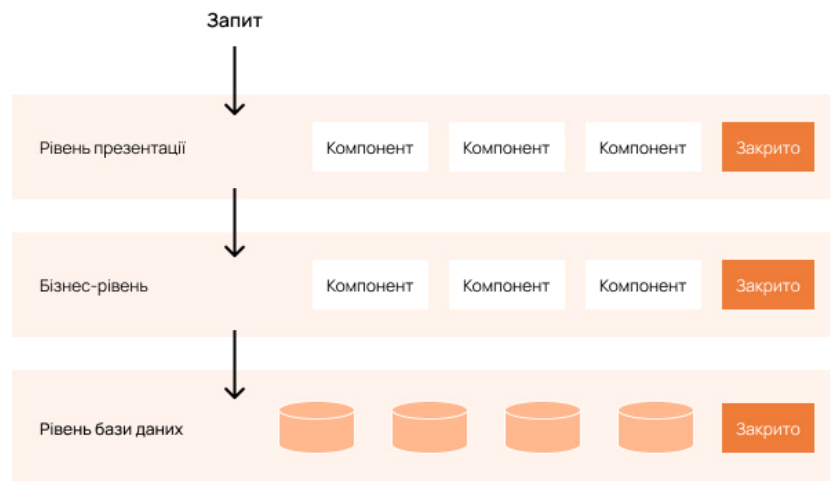


Рисунок 2.10 – Типова схема трьохшарової архітектури

Розроблювані мікросервіси побудовані за принципами трьохшарової архітектури, яка включає в себе наступні основні шари: презентаційний (рівень презентації), бізнес-рівень та шар бази даних.

– Презентаційний шар. Цей шар відповідає за взаємодію з кінцевим користувачем. Він включає в себе графічний інтерфейс користувача (GUI), який може бути створено за допомогою сучасних веб-технологій. Інтерфейс є інтуїтивно зрозумілим та зручним для користувача, він включає в себе різні форми для пошуку, перегляду інформації предметну область та комунікації з іншими користувачами. Може описувати правила та формати, за якими дані повинні передаватись від клієнта до сервіса, та навпаки.

– Бізнес-рівень. Цей шар включає в себе основні алгоритми та процедури обробки даних. Він відповідає за обробку запитів від користувача, виконання необхідних операцій та відправку відповіді до презентаційного шару.

– Шар бази даних. Цей шар відповідає за зберігання та доступ до даних. Він включає в себе базу даних. Тут зберігаються класи-сутності, які представлені у базах даних у вигляді таблиць. Також шар може містити інтерфейси, які описують методи взаємодії з даними для шару бізнес логіки.

Схему взаємодії між сервісами зображено на рисунку 2.11.

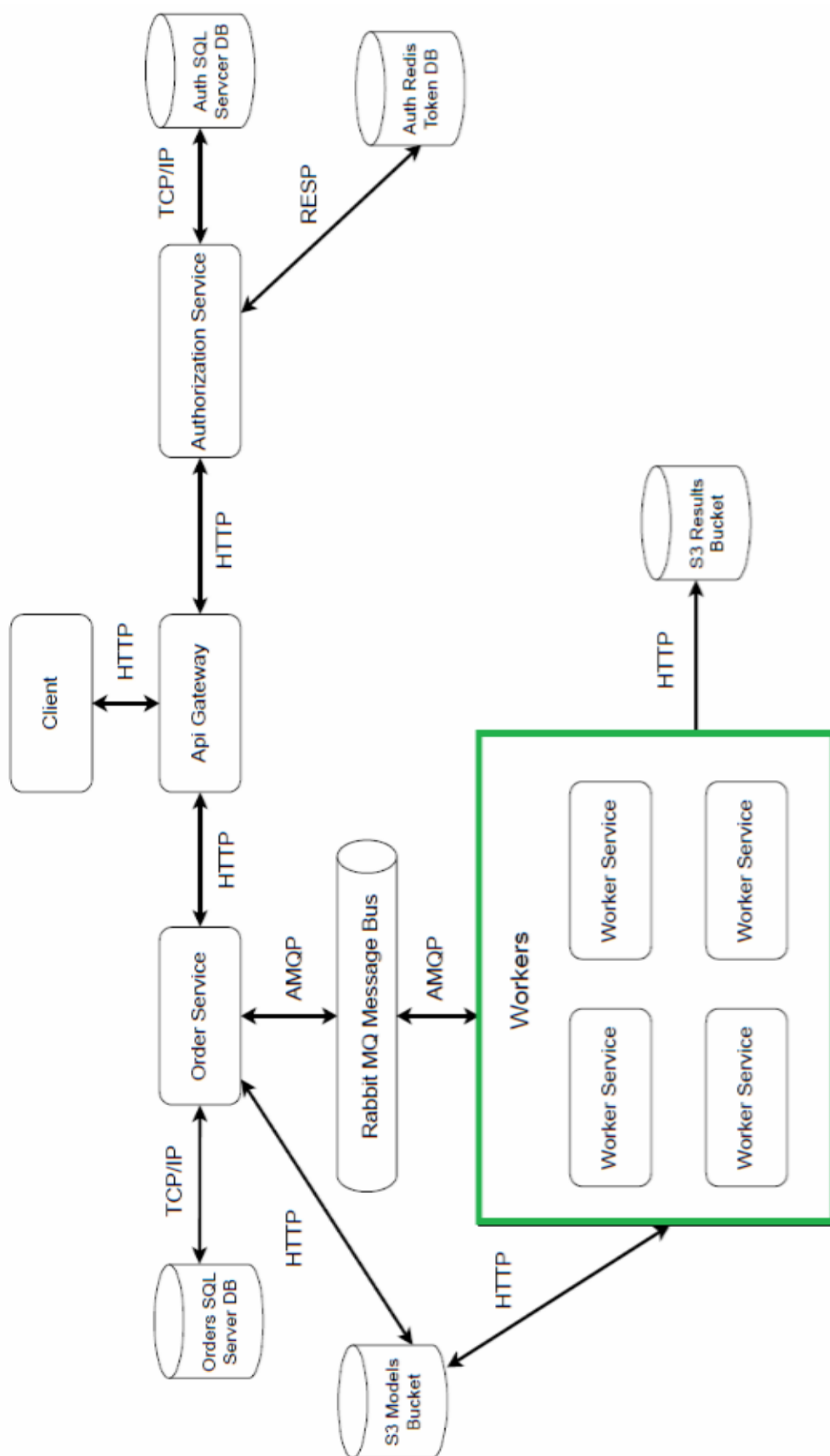


Рисунок 2.11 - Схема взаємодії між сервісами

Для додавання пакетів до проектів на платформі .NET використовується менеджер пакетів Nuget. Він містить у собі великий спектр різних пакетів та надає можливості для версіонування встановлених пакетів. Список використаних пакетів, бібліотек та наборів інструментів зазначений нижче:

- Microsoft.Extensions.Options - додає DI механізм для налаштувань проекту
- RabbitMQ.Client - бібліотека, що містить клієнт брокера повідомлень RabbitMQ
- SkiaSharp - Бібліотека для створення зображень на основі біт мапи
- Microsoft.Extensions.Hosting - бібліотека для створення бекграунд-процесів
- Microsoft.Extensions.Logging - додає можливість логування інформації
- Swashbuckle.AspNetCore - ібліотека для генерування Swagger документації для API
- Microsoft.EntityFrameworkCore.Relational - додає можливості гнучкого налаштування відношень між сутностями у базі даних
- StackExchange.Redis - клієнт для роботи із СУБД Redis
- AutoMapper - інструмент для мапінгу об'єктів між різними шарами архітектури
- Isopoh.Cryptography.Argon2 - бібліотека для хешування паролів за допомогою алгоритму Argon2
- Microsoft.EntityFrameworkCore - являє собою ORM для спрощення роботи із базами даних;
- Microsoft.IdentityModel.Tokens - бібліотека, що надає криптографічні методи для шифрування токенів доступу
- Microsoft.EntityFrameworkCore.SqlServer - додає DI механізм для створення підключення до СУБД Microsoft SQL Server

					ДРБ.ІІ – 29.00.00.000 ІІЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

- AWSSDK.Core - SDK для роботи із клієнтом AWS
- Microsoft.IdentityModel.Tokens.Jwt - бібліотека для створення та валідації JWT
- Newtonsoft.Json - фреймворк для роботи з файлами формату JSON
- AutoMapper.Extensions.Microsoft.DependencyInjection - додає DI механізм для AutoMapper
- AWSSDK.S3 - SDK для роботи із сервісом S3.

Отже, в даному розділі виконано аналіз вимог, моделювання та розробка структури розроблюваної системи графічного моделювання.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		59

3. РОЗРОБКА АРХІТЕКТУРИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ГРАФІЧНОГО МОДЕЛЮВАННЯ

3.1. Опис архітектури сервісу замовлень

CloudStorageService – клас-сервіс, який відповідає за роботу з хмарним сховищем S3. Нижче наведено опис методів класу CloudStorageService:

- UploadFileAsync - равантажує файл на хмарне сховище S3;
- DeleteFileAsync - видаляє файл по вказаному посиланню із хмарного сховища;
- GetObjectStreamAsync - зчитує файл за вказаним посиланням у вигляді потоку байтів.

OrdersProducer – клас-сервіс, який відповідає створення та відправку повідомлень про створення нового замовлення у брокер повідомлень RabbitMQ. Клас OrdersProducer містить метод PushMessage який створює та надсилає повідомлення про створення нового замовлення до брокеру повідомлень.

OrderMessage – це клас повідомлення, яким обмінюються між собою сервіси. Являє собою модель замовлення. На рисунку 3.1 показано клас OrderMessage.

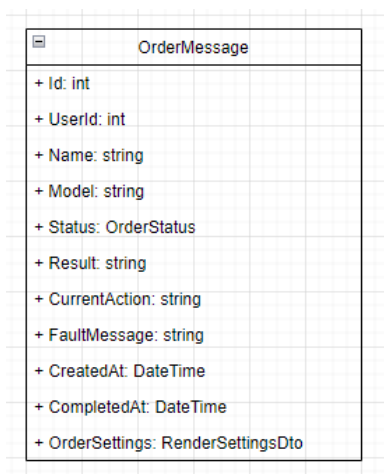


Рисунок 3.1 - Клас OrderMessage

Клас OrderMessage містить наступні поля:

- Id - ідентифікатор замовлення;
- UserId - ідентифікатор користувача, що створив замовлення;
- Name - ім'я замовлення;
- Model - посилання на файл моделі у хмарному сховищі;
- Status - поточний статус замовлення;
- Result - посилання на файл результуючого зображення у хмарному сховищі;
- CurrentAction - поточна дія, що виконує воркер над замовленням;
- FaultMessage – [арактеризує помилку, що сталася під час рендерингу;
- CreatedAt - дата створення замовлення;
- CompletedAt - дата завершення замовлення;
- OrderSettings - налаштування замовлення.

OrderSettingsDto – клас, що агрегує у собі всі налаштування замовлення. На рисунку 3.2 зображено описано класи налаштувань замовлення.

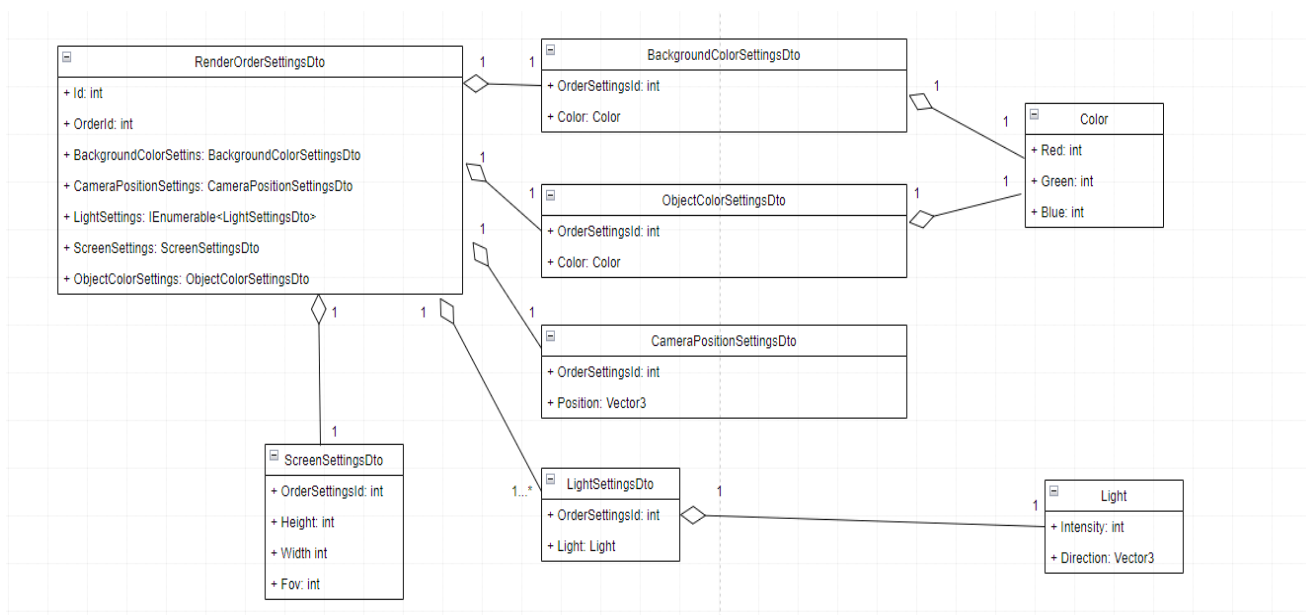


Рисунок 3.2 - Опис класів налаштування замовлень

Розглянемо опис архітектури воркера. JpegImageWriter – клас-сервіс, що відповідає за створення зображень формату .JPEG за допомогою бібліотеки SkiaSharp. Клас JpegImageWorker містить метод WriteImage який перетворює векторну бітмапу на зображення формату .JPEG.

Tree – це клас, що являє собою структуру даних octree. Кожний вузол даного дерева представлений класом TreeNode. Вузли складається із восьми дочірніх вузлів та просторового кубу. Просторові куби задаються класом Cube, який зберігає у собі вісім точок у просторі, а також містить логіку перевірки, чи перетинається від заданим променем. Клас Tree містить наступні методи:

- Initialize - створює та ініціалізує вузли дерева;
- FindIntersections - проводить рекурсивний обхід дерева в пошуку полігонів, які перетинаються з променем починаючи з кореня;
- Rebuild - ділить куб у вузлі на вісім однакових менших кубів, створює дочірні вузли та розподіляє полігони між ними;
- FindIntersections - рекурсивно перевіряє перетин променю для кожного з дочірніх вузлів;
- PositionFinder - визначає індекс дочірнього вузла, якому належить введена точка, внаслідок чого отримується індекс дочірнього вузла у списку дочірніх вузлів.

Renderer – клас-сервіс, що рендерить об'ємну модель і містить наступні методи:

- GetCameraDirection - обчислює вектор напрямку камери відносно конкретного пікселя та налаштувань екрану чим визначає вектор напрямку камери;
- Initialize - фарбує усі пікселі у заданий в налаштуваннях колір;
- CastRay - пускає промінь у вибраному напрямку та обчислює значення кольору для заданих налаштувань світла. Якщо промінь перетинає

полігон, то повертає точку перетину та колів. У протилежному випадку – нічого;

- ProcessPixel - обчислює значення кольору для конкретного пікселя у результуючому зображенні.

RenderService – клас-сервіс, який запускає ланцюжок процесу рендерингу від зчитування моделі, до завантаження результуючого зображення до хмарного сховища.

3.2. Конструювання системи графічного моделювання згідно принципів REST

В рамках даного проекту для презентаційного шару кожного мікросервіса являє собою API, побудоване за принципами REST. Кожен ендпоінт API являє собою окремий метод певного контролера.

Використання REST (Representational State Transfer) архітектури в розробці програмного забезпечення має численні переваги. Нижче наведено основні з них:

1. Масштабованість:

- Горизонтальна масштабованість: Легкість масштабування сервісів через додавання нових серверів без змін у клієнтському коді.
- Збалансоване навантаження: RESTful API можуть ефективно використовувати балансувальники навантаження для розподілу запитів.

2. Простота та легкість використання:

- Простий синтаксис: Використання стандартних HTTP методів (GET, POST, PUT, DELETE), що зрозумілі і легкі у використанні.
- Зрозумілі URL: Використання уніфікованих ресурсних ідентифікаторів (URI), які легко читати та інтерпретувати.

3. Клієнт-серверна незалежність:

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		63

- Сервер та клієнт розділені: Розподіл обов'язків між клієнтом та сервером, що дозволяє розвивати їх незалежно один від одного.

- Заміна компонентів: Можливість заміни або оновлення клієнта або сервера без впливу на іншу частину системи.

4. Кешування:

- Підтримка кешування: Використання HTTP заголовків для керування кешуванням відповідей, що знижує навантаження на сервер та покращує продуктивність.

- Ефективне використання ресурсів: Зменшення кількості запитів до сервера завдяки використанню кешованих даних.

5. Стандартизація:

- Використання стандартних протоколів: REST побудований на базі стандартних протоколів (HTTP, HTTPS), що робить його сумісним з більшістю систем.

- Уніфіковані методи: Використання стандартних методів CRUD (Create, Read, Update, Delete) для управління ресурсами.

6. Гнучкість:

- Підтримка різних форматів даних: Можливість використання різних форматів представлення даних (JSON, XML, HTML).

- Різні типи клієнтів: RESTful API можуть бути споживані різними типами клієнтів, такими як веб-додатки, мобільні додатки, IoT пристрої.

7. Модульність та повторне використання:

- Модульність: Легкість поділу на модулі та повторного використання компонентів.

- Зручне розширення: Додавання нових функцій без зміни існуючих компонентів.

8. Легкість тестування:

- Тестування через HTTP: Простота тестування API через стандартні HTTP клієнти (наприклад, Postman, curl).

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

- Аналіз та дебагінг: Легкість аналізу запитів та відповідей завдяки використанню стандартних HTTP кодів стану.

9. Висока доступність та надійність:

- Мікросервісна архітектура: Підтримка мікросервісної архітектури, що підвищує надійність та доступність сервісів через розподіл навантаження.

10. Зручність інтеграції:

- API як контракт: Визначення чітких контрактів між клієнтом та сервером через API, що спрощує інтеграцію з іншими системами.

- Взаємодія з різними мовами: Можливість взаємодії з клієнтами, написаними на різних мовах програмування.

Всі ці переваги роблять REST архітектуру популярним вибором для розробки веб-сервісів та API, забезпечуючи високу продуктивність, гнучкість та зручність у використанні.

Типова схема REST API зображена на рисунку 3.3.

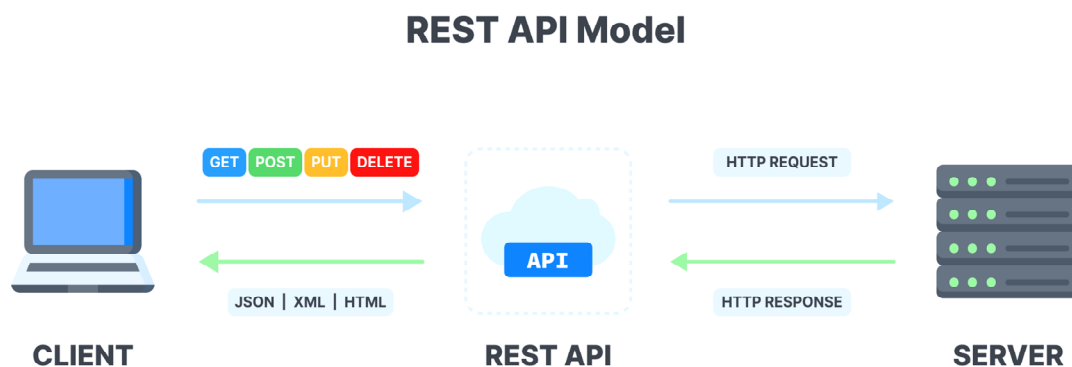


Рисунок 3.3 – Типова модель REST API

Щоб відповідати архітектурі REST, API повинно відповідати наступним вимогам:

- повинно використовувати клієнт-серверну модель;
- стан клієнта не повинен зберігатися на стороні сервера;
- запити повинні кешуватися;

– інтерфейс повинен бути одноманітним та бути побудованим навколо «ресурсів», тобто абстракцій, які формують можливості інтерфейсу.

3.3. Розробка інтерфейсу користувача системи графічного моделювання та рендерингу

На рисунку 3.4 зображено реєстрацію нового користувача в розробленій системі.

Sign in Sign up

Email: test@gmail.com

Password: *****

Sign-up

Рисунок 3.4 – Форма реєстрації нового користувача

Наступним кроком є створення нового замовлення на рендеринг. Для цього необхідно перейти до вкладки «Create order» та ввести параметри для замовлення. На рисунку 3.5 зображено вікно створення нового замовлення на рендеринг.

My orders Create order Logged as test@gmail.com

Name: CowOrder

Model: cow.obj

Screen height: 600

Screen width: 600

FOV: 150

Camera position: 0 2 0

Background color: [dropdown]

Object color: [dropdown]

Lights: +

Intensity: 3 Position X: 0 Position Y: 2 Position Z: 0

Create

Рисунок 3.5 – Створення нового замовлення на рендеринг

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

У якості об'ємної моделі було обрано файл cow.obj, який являє собою модель корови. На рисунку 3.6 зображено модель cow.obj у додатку Paint 3D.

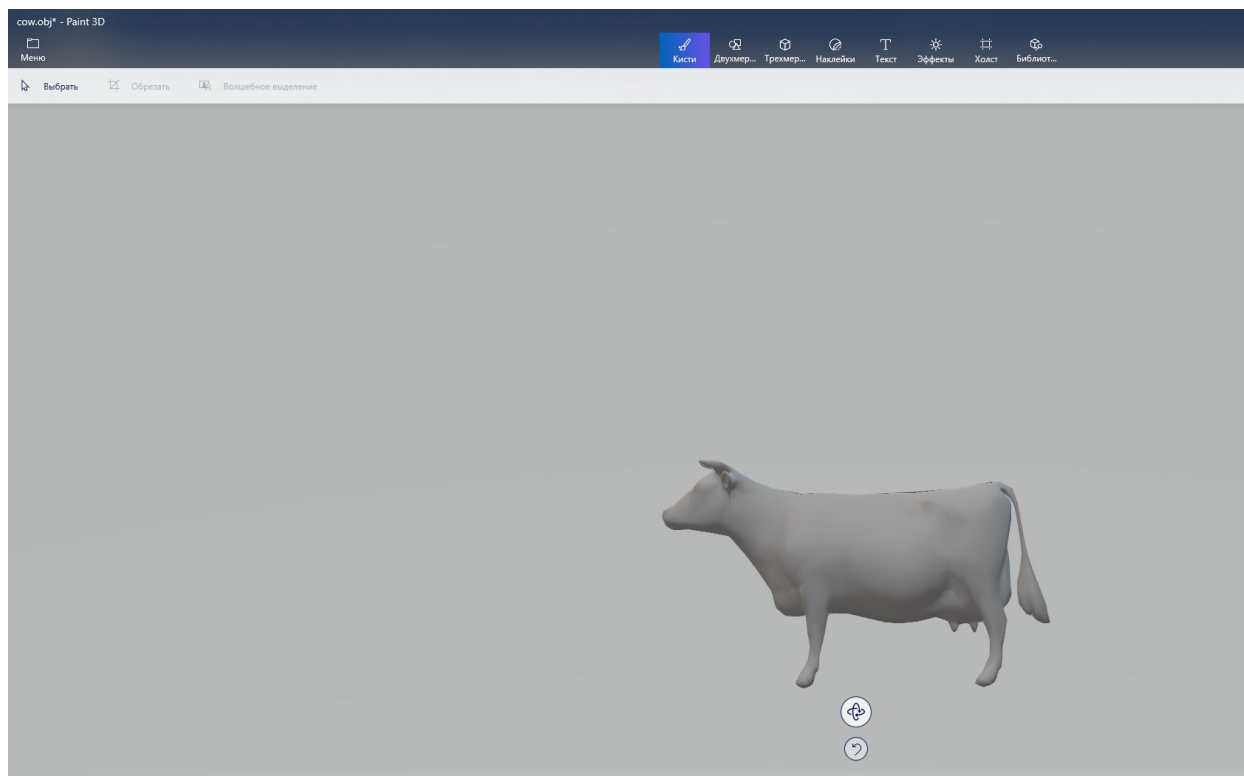


Рисунок 3.6 – Вигляд моделі об'єкта у додатку Paint 3D

Нове замовлення можна побачити у вкладці «My orders». У колонці «Status» можна побачити зміну стану замовлення у режимі реального часу. Початковим станом є стан «Created». Коли замовлення починає оброблятися, його стан змінюється на «InProgress», а у колонці «Current action» можна побачити поточний етап виконання. Коли замовлення виконано, його стан змінюється на «Completed» та у колонці «Result» з'являється посилання на результуюче зображення.

На рисунках 3.7 – 3.9 зображено зміни стану замовлення виконання рендерингу.

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

Id	Name	Status	Current action	ErrorMessage	Result	Created	Completed
46	CowOrder3	Created				06.06.2024 20:34:01	01.01.0001 0:00:00

Рисунок 3.7 – Початковий стан замовлення

Id	Name	Status	Current action	ErrorMessage	Result	Created	Completed
46	CowOrder3	InProgress	Rendering model			06.06.2024 20:34:01	01.01.0001 0:00:00

Рисунок 3.8 – Стан замовлення в статусі «In progress»

Id	Name	Status	Current action	ErrorMessage	Result	Created	Completed
46	CowOrder3	Completed			https://render-service-results.s3.a	06.06.2024 20:34:01	06.06.2024 20:34:41

Рисунок 3.9 – Стан замовлення в статусі «Completed»

У випадку використання замовлення на рендеринг із некоректною моделлю отримаємо наступну помилку, що показана на рисунку 3.10. Некоректною моделлю вважається модель у вигляді пустого файлу, але із налаштуванням екрану, позиції камери, кольорів об'єкта та заднього отримуємо статус Faulted з відповідним повідомленням про помилку зчитування моделі.

Id	Name	Status	Current action	ErrorMessage	Result	Created	Completed
43	EmptyOrder	Faulted		Invalid model		06.06.2024 19:33:29	06.06.2024 19:33:29

Рисунок 3.10 – Стан замовлення в статусі «Faulted»

Також користувач має можливість перегляду власних замовлень на рендеринг (рис. 3.11).

My orders

Create order

Logged as test@gmail.com

Refresh

id	Name	Status	Current acticErrorMessage	Result	Created	Completed
44	CowOrder	Completed		https://render-service	06.06.2024 20:13:45	06.06.2024 20:13:53
45	CowOrder2	Completed		https://render-service	06.06.2024 20:16:51	06.06.2024 20:16:59

Рисунок 3.11 – Список замовлень користувача

При натисканні на посилання у колонці «Result», відкривається браузер з відповідним посиланням. На рисунку 3.10 зображено результат рендерингу.

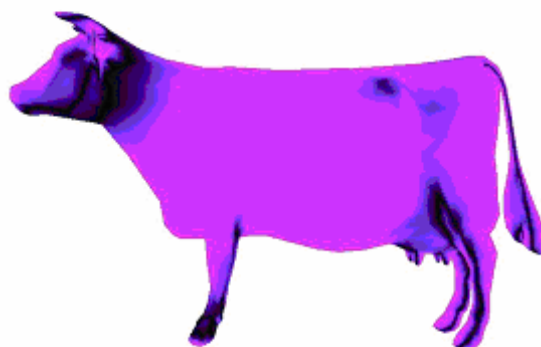


Рисунок 3.12 – Результат виконання рендерингу

На рисунках 3.13 і 3.14 показано інші результати роботи системи графічного моделювання та рендерингу, початкового зображення і кінцевого відповідно.



Рисунок 3.13 – Вигляд моделі об’єкта (початкове зображення)

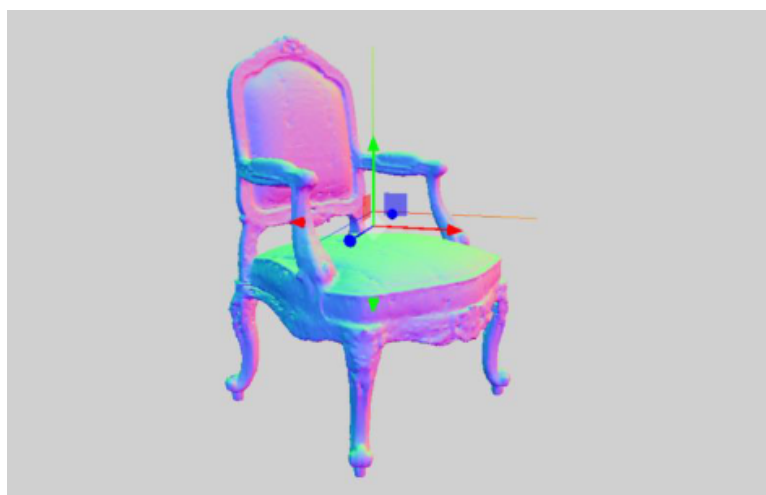


Рисунок 3.14 – Результат виконання рендерингу

3.4. Виконання розгортання програмних сервісів

Розгортання усіх сервісів програмних засобів для тестового середовища було виконано на платформі AWS.

Amazon Web Services (AWS) – це масштабована та гнучка платформа хмарних обчислень, розроблена компанією Amazon. AWS надає широкий спектр послуг, які дозволяють організаціям орендувати обчислювальні потужності, зберігання даних, бази даних та інші ресурси в Інтернеті замість придбання та обслуговування власної фізичної інфраструктури. AWS надає масштабовані та економічно вигідні рішення для

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

різних потреб бізнесу, від стартапів до великих корпорацій, забезпечуючи високу доступність, надійність та безпеку.

Основні особливості та компоненти платформи AWS включають:

- Обчислювальні ресурси;
- Зберігання даних та бази даних;
- Інструменти розробки та DevOps;
- Інструменти управління та моніторингу.

Для розгортання програмного забезпечення була використана система CI/CD GitHub Actions. Дана система дозволяє створювати робочі процеси, які виконуються кожен раз, коли у репозиторії виникають певні зміни. В даному випадку такими змінами є коміти, які додаються у головну гілку репозиторію. Робочий процес являє собою послідовність команд, або робіт, які записуються у файл формату `.yaml`.

На рисунках 3.15 – 3.17 зображено статус виконання останніх збірок для кожного із сервісів.

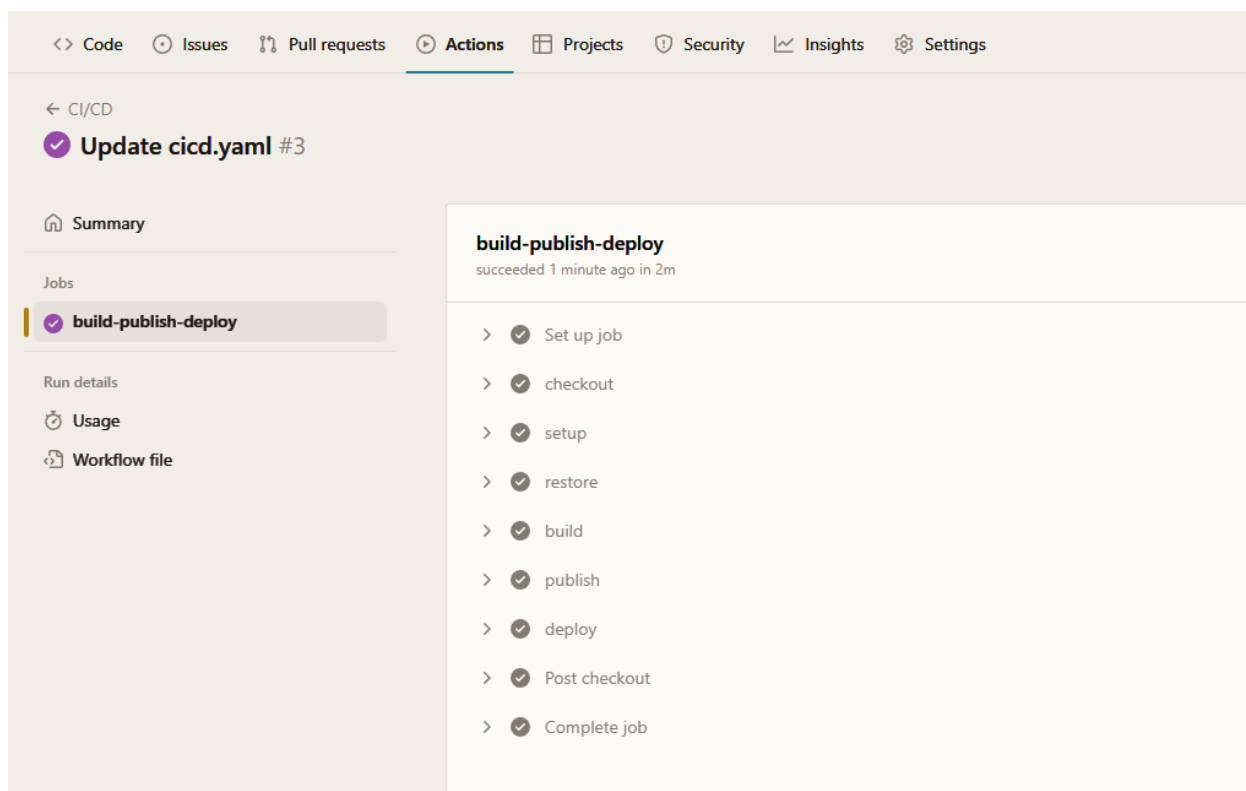


Рисунок 3.15 – Статус виконання останньої збірки для сервісу авторизації

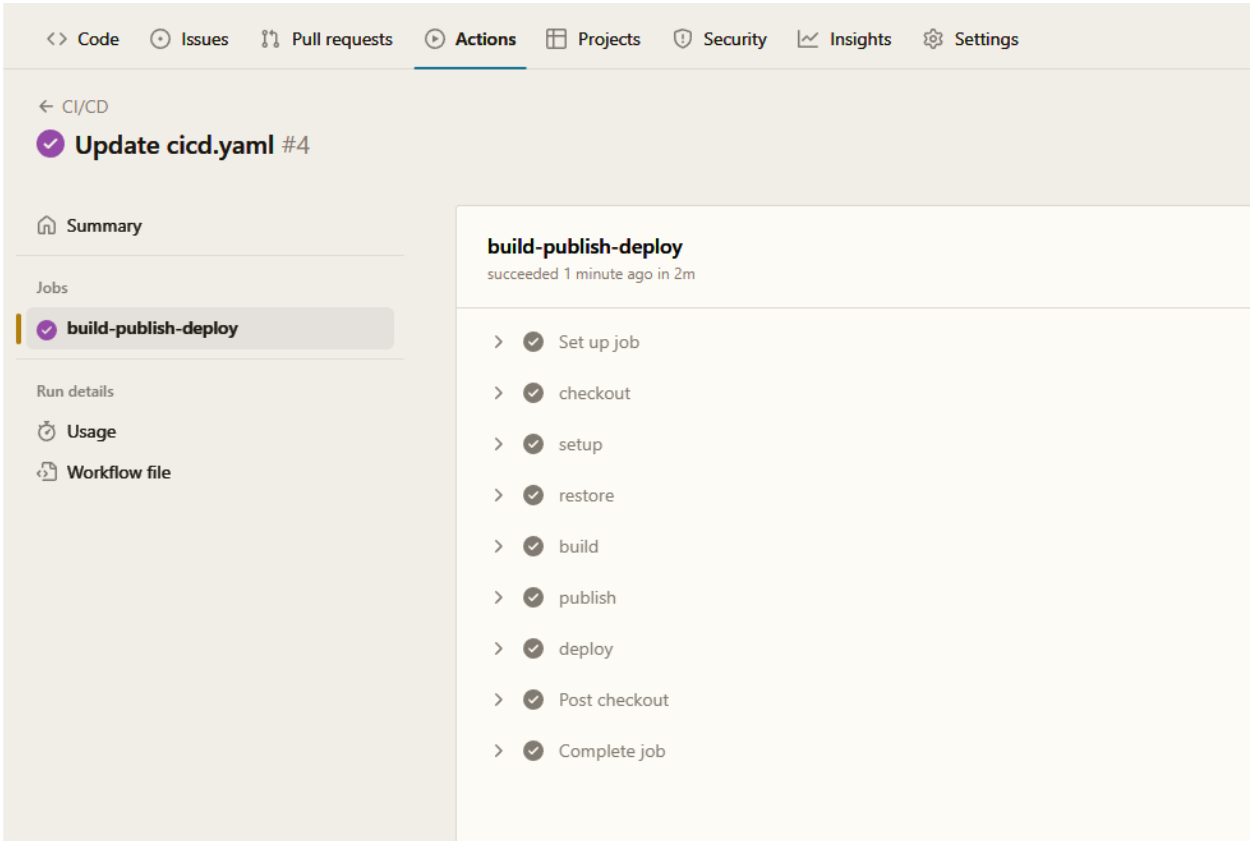


Рисунок 3.16 – Статус виконання останньої збірки для сервісу замовлень

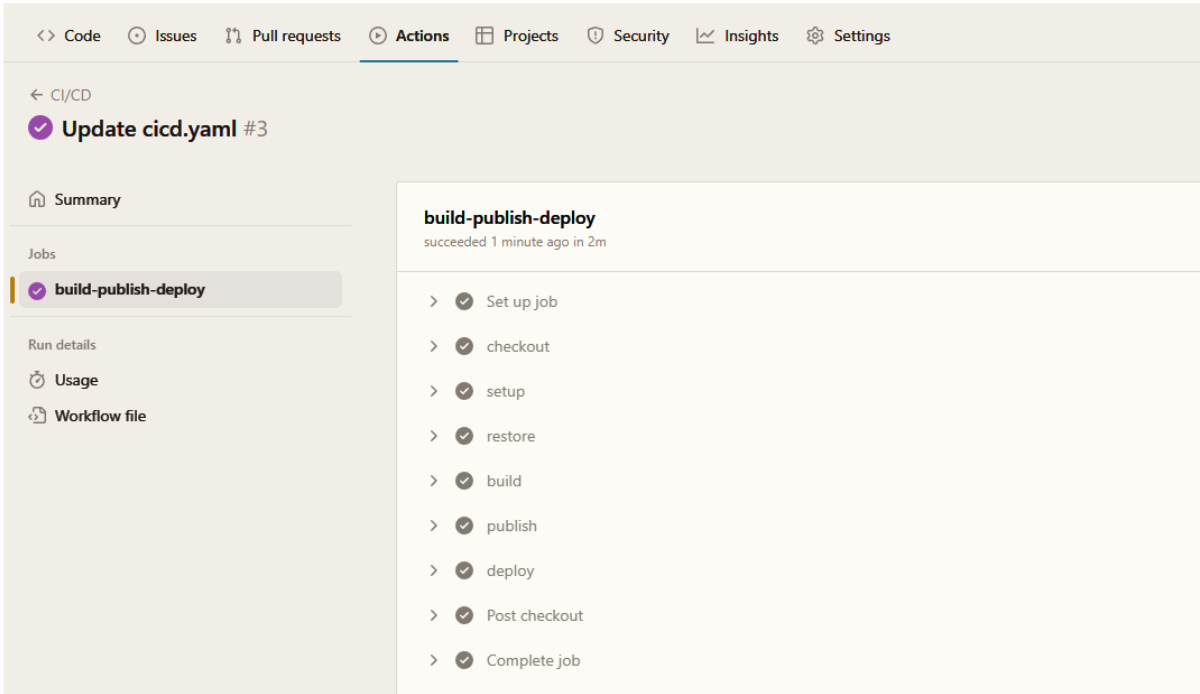


Рисунок 3.17 – Статус виконання останньої зборки для воркєру

Для кожного репозиторію виконується наступний список команд:

- отримання вихідного коду з гілки master і завантаження пакетів з Nuget;
- збірка рішення та docker-образу з подальшою публікацією docker-образу у Docker Hub;
- виконання розгортання на платформі AWS.

Отже, було представлено архітектуру програмної системи графічного моделювання і рендерингу, показано інтерфейс системи, функціонал створення та перегляду замовлень на рендеринг. Також було наведено список ресурсів, що було створено під час розгортання проекту на платформі AWS.

					ДРБ.ІІ – 29.00.00.000 ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

В дипломній роботі проведена реалізація програмного забезпечення системи графічного моделювання. Проведено аналіз існуючих рішень та предметної області побудови систем графічного моделювання та рендерингу, проаналізовані основні сучасні розробки, визначено переваги та недоліки цих рішень. В результаті проведеного аналізу сформульована постановка задачі, наведене призначення, цілі та задачі розробки.

Основною метою розробки є надання можливості виконання рендерингу об'ємних моделей за рахунок виконання процесу з будь-якого пристрою з доступом до мережі Інтернет. В роботі описано предметну область, проаналізовано відомі технічні рішення, проведено порівняння з аналічними системами, створено діаграму варіантів використання, наведені функціональні та нефункціональні вимоги до програмної системи графічного моделювання.

Створено BPMN діаграми та здійснено опис основного бізнес-процесу, обґрунтовано архітектурне рішення, побудово схеми взаємодії між сервісами, наведено функціонал кожного з мікросервісів. Також здійснено опис принципів, за якими було побудовано API кожного з сервісів, наведено опис використаних утиліт, бази даних і таблиць кожного сервісу.

Виконано опис процесів і методи тестування, наведено список утиліт, що використовуються для аналізу якості коду, та наведено інтерфейс розробленої системи графічного моделювання. Також проведено опис процесів розгортання та безперервного супроводу програмної системи.

Результатом роботи є прототип системи, який демонструє можливість ефективного графічного моделювання та рендерингу засобами веб, що відкриває нові перспективи для розробки та застосування веб-додатків у галузі комп'ютерної графіки та дизайну.

					ДРБ.ІІ – 29.00.00.000 ІЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Real-time Rendering of Complex Photorealistic Landscapes Using Hybrid Level-of-Detail Approaches – 2005: https://www.researchgate.net/publication/30012777_Realtime_Rendering_of_Complex_Photorealistic_Landscapes_Using_Hybrid_Level-of-Detail_Approaches
2. ReSharper Documentation – 2022: <https://www.jetbrains.com/resharper/documentation/documentation.html>
3. Інформація що до налаштування corona render <https://www.digitalmediaworld.tv/vfx/1618-chaos-group-acquires-render-legion-and-corona-renderer>
4. Приклади роботи corona render <https://80.lv/articles/chaos-corona-8-for-3Ds-max-and-cinema-4d-released>
5. Richardson,Chris Microservice Patterns [Text] / Chris Richardson. – Fall, 2017. – 375 p.
6. 3D object manipulation in a single photograph using stock 3D models / N. Kholgade та ін. ACM Transactions on Graphics (TOG). 2014. Т. 33. С. 1–12.
7. Compressible-composable NeRF via Rank-residual Decomposition / J. Tang та ін. ArXiv. 2022. Abs/2205.14870.
8. Editing Conditional Radiance Fields / S. Liu та ін. Proceedings of the International Conference on Computer Vision (ICCV). 2021
9. What Is The Difference Between Smooth Shading And Flat Shading? – 2022. – <https://www.blenderbasecamp.com/home/what-is-the-difference-between-smooth-shading-and-flat-shading/>
- 10.T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire, Real-Time Rendering, Fourth Edition. 2015
- 11.Ángel Ortiz. A Primer On Efficient Rendering Algorithms & Clustered Shading: <http://www.aortiz.me/2018/12/21/CG.html>.

					ДРБ.ІІ – 29.00.00.000 ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		75

12. Nvidia. Depth precision visualized.: <https://developer.nvidia.com/content/depth-precision-visualized>.
13. Adelson E. H., Bergen J. R. The plenoptic function and the elements of early vision. Computational models of visual processing. 1991. T. 1, № 2. С. 3 – 20.
14. Alexander Overvooorde. Vulkan Tutorial: <https://vulkan-tutorial.com>.
15. Khronos Group. Vulkan 1.3.237 – A specification: <https://registry.khronos.org/vulkan/specs/1.3-extensions/html/vkspec.html>.
16. D. Zhdan, Nvidia. Tiled shading: light-culling – reaching the speed of light, GDC 2018: https://ubm-twvideo01.s3.amazonaws.com/o1/vault/gdc2016/Presentations/Zhdan_Sjoholm_Light_culling_MGPU.pdf.
17. J. Turánszki, Optimizing tile-based light culling: <https://wickedengine.net/2018/01/10/optimizing-tile-based-light-culling/>.
18. An overview of rendering techniques: <https://www.sciencedirect.com/science/article/abs/pii/S0097849390900140>
19. Fast Ray-Triangle Intersections by Coordinate Transformation. – 2016: <https://jcgt.org/published/0005/03/03/>
20. McMillan L., Bishop G. Plenoptic modeling: an image-based rendering system. Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. 1995.
21. M. Kartashov, V. N. Oleynikov, S. A. Sheyko, I. V. Koryttsev, S. I. Babkin, O. V. Zubkov, "Peculiarities of small unmanned aerial vehicles detection and recognition," Telecommunications and Radio Engineering, 2019, V. 78, Iss. 9, pp. 771–781. DOI: 10.1615/TelecomRadEng.v78.i9.30.
22. V. N. Oleynikov, O. V. Zubkov, V. M. Kartashov, I. V. Koryttsev, S. I. Babkin, S. A. Sheiko, "Investigation of detection and recognition efficiency of small unmanned aerial vehicles on their acoustic radiation," Telecommunications and Radio Engineering, 2019, V. 78, Iss. 9, pp. 759–770. DOI: 10.1615/TelecomRadEng.v78.i9.20

					ДРБ.ІІ – 29.00.00.000 ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		76

23. Microservices // Wikipedia. The free encyclopedia. – Access mode: <https://en.wikipedia.org/wiki/Microservices>.
24. What is Microservices Architecture? // Smartbear. – Access mode: <https://smartbear.com/learn/api-design/what-are-microservices/>.
25. Non-Photorealistic Rendering. – URL: https://www.researchgate.net/publication/236973460_Non-Photorealistic_Rendering
26. T. A. Funkhouser and C. H. Sequin. Адаптивний алгоритм відображення для інтерактивних частот кадрів під час візуалізації складних віртуальних середовищ. SIGGRAPH, с. 247–254, (1999).
27. Sch'ounerger J. L., Frahm J.-M. Structure-from-Motion Revisited. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. С. 4104–4113.
28. Implicit neural representations with periodic activation functions / V. Sitzmann та ін. Advances in Neural Information Processing Systems. 2022. Т. 33. С. 7462–7473.
29. Hesina and D. Schmalstieg. Мережева архітектура для віддаленого візуалізації. Матеріали другого міжнародного практикуму з розподіленого інтерактивного моделювання та додатків у реальному часі, с. 88–91, (1998).
30. PlenOctrees for Real-time Rendering of Neural Radiance Fields / A.Yu та ін. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). 2021. С. 5732–5741
31. MS SQL server history and advantages. – 2014: <https://bytescout.com/blog/2014/09/ms-sql-server-history-and-advantages.html>

ДОДАТКИ

Додаток А

Лістинг програмних кодів

AuthorizationService.cs

```
public class AuthorizationService : IAuthorizationService {
    private readonly ITokenManager manager;
    private readonly IRefreshTokenService refreshTokenService; private
    readonly IUserService userService; private readonly IPasswordService
    passwordService;
    public AuthorizationService(ITokenManager manager,
    IRefreshTokenService tokenService,
    IUserService userService,
    IPasswordService passwordService)
    {
        manager = manager;
        refreshTokenService = tokenService; userService = userService;
        passwordService = passwordService;
        public async Task<ProfileDto> Authorize(CredentialsDto credentials)
        {
            UserDto user
            try { '
            user = await userService.GetUserByEmail(credentials.Email);
            }
            catch (EntityNotFoundException)
            {
                throw new WrongCredentialsException($"No such email
                {credentials.Email}");
            }
            var isPasswordValid = passwordService.Verify(new HashedPassword()
            {
                PasswordHash = user.PasswordHash,
                Salt = user.PasswordSalt }, credentials.Password);
            if (!isPasswordValid)
            {
                throw new WrongCredentialsException($"Wrong password for email {
                credentials.Email}");
            }
        }
    }
}
```

```

var tokens = manager.GenerateTokens(new JwtPayload() {
    Userid = user.Id,
    Claims = new List<Claim>()
    {
        new(JwtRegisteredClaimNames.Email, user.Email), new("id",
        user.Id.ToString())
    }
});
await refreshTokenService.Update(tokens.RefreshToken);
var profile = new ProfileDto()
{
    Token = tokens, User = user
return profile;
}
}

```

RedisEntity.cs

```

public class RedisEntity : IEntity<RedisKey> {
    public RedisKey Id { get; set; } public RedisValue Value { get; set; }
}

```

User.cs

```

public class User : EntityBase {
    public string Email { get; set; }
    public string PasswordHash { get; set; }
    public string PasswordSalt { get; set; }
    public string PasswordVer { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime LastVisit { get; set; }
    public DateTime LastEdit { get; set; }
    public DateTime LastPasswordReset { get; set; }
    public IEnumerable<Role> Roles { get; set; }
}

```

IEntity.cs

```

public interface IEntity<TKey> where TKey {
    TKey Id { get; set; }
}

```

IRedisRepository.es

```

public interface IRedisRepository<TEntity, in TKey> where TEntity :
    IEntity<TKey> where

```

```

TKey : IEquatable<TKey>
{
Task Delete(TEntity entity); Task<TEntity> Update(TEntity entity);
Task<TEntity> Insert(TEntity entity); Task<TEntity> GetByIdAsync(TKey
id);

```

Файл IRedisUnitOfWork.cs

```

public interface IRedisUnitOfWork : IDisposable {
IRedisRepository<RedisEntity, RedisKey> Repository { get; }
}

```

IRepository.cs

```

public interface IRepository<TEntity, in TKey> : IDisposable where
TEntity : IEntity<TKey> where
TKey : IEquatable<TKey>
{
Task Delete(TEntity entity); Task<TEntity> Update(TEntity entity);
Task<TEntity> Insert(TEntity entity); Task<TEntity> GetByIdAsync(TKey
id); IQueryable<TEntity> GetAll(); Task<IEnumerable<TEntity>>
GetAllAsync()
}

```

IUnitOfWork.cs

```

public interface IUnitOfWork : IDisposable {
IRepository<User, int> UsersRepository { get; }
IRepository<Role, int> RolesRepository { get; }
Task SaveAsync();
}

```

RedisRepository.cs

```

public class RedisRepository : IRedisRepository<RedisEntity, RedisKey>
{
private readonly IConnectionMultiplexer redis; public
RedisRepository(IConnectionMultiplexer redis)
{
redis = redis;
public Task Delete(RedisEntity entity)
{
var db = redis.GetDatabase();
return db.StringGetDeleteAsync(entity.Id);
public async Task<RedisEntity> Update(RedisEntity entity) {
var db = redis.GetDatabase();

```

```

await db.StringSetAsync(entity.Id, entity.Value); return entity;
public async Task<RedisEntity> Insert(RedisEntity entity)
{
var db = redis.GetDatabase();
var res = await db.StringSetAsync(entity.Id, entity.Value, when:
When.NotExists);
return res ? entity :
throw new EntityAlreadyExistsException($"Pair with key {entity.Id}
already exists");
} '
public async Task<RedisEntity> GetByIdAsync(RedisKey id) {
var db = redis.GetDatabase ();
var value = await db.StringGetAsync(new RedisKey(id)); if (value ==
default)
{
throw new EntityNotFoundException($"Value with key {id} not
return new RedisEntity {
Id = id,
Value = value
};
}
}
}

```

RedisUnitOfWork.cs

```

public class RedisUnitOfWork : IRedisUnitOfWork {
private readonly IConnectionMultiplexer redis; private bool disposed;
public IRedisRepository<RedisEntity, RedisKey> Repository { get;
private set; }
public RedisUnitOfWork(IConnectionMultiplexer redis)
{
redis = redis;
Repository = new RedisRepository( redis);
}
public void Dispose()
{
Dispose(true);
GC.SuppressFinalize(this);
}
protected virtual void Dispose(bool disposing) {

```

```

if (! disposed)
{
    if (disposing)
    {
        redis.Dispose ();
    }
    disposed = true;
} }

```

UnitOfWork.cs

```

public class UnitOfWork : IUnitOfWork {
    private readonly DbContext context; private bool disposed = false;
    private IRepository<User, int> usersRepository; private
    IRepository<Role, int> rolesRepository;
    public IRepository<User, int> UsersRepository => usersRepository ??=
    new Repository<User, int>( context);
    public IRepository<Role, int> RolesRepository => rolesRepository ??=
    new Repository<Role, int>( context);
    public UnitOfWork(UserDbContext context)
    {
        context = context;
    }
    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }
    public Task SaveAsync()
    protected virtual void Dispose(bool disposing)
    {
        if (Ithis.disposed) {
            if (disposing) {
                context.Dispose ();
            }

            }
        this.disposed = true;
    }
}

```

RegistrationService.es

```

public class RegistrationService : IRegistrationService {

```

```

private readonly IUserService userService; private readonly
IPasswordService passwordService; private readonly
IRefreshTokenService tokenService; private readonly ITokenManager
tokenManager;
public RegistrationService(IUserService userService, IPasswordService
passwordService, IRefreshTokenService tokenService,
ITokenManager tokenManager)
{
userService = userService; passwordService = passwordService;
tokenService = tokenService; tokenManager = tokenManager;
public async Task<ProfileDto> RegisterUser(UserDto user)
{
var isEmailNotExists = await ValidateUserEmail(user.Email); if
(!isEmailNotExists)
{
throw new UserAlreadyExistsException(user.Email);
} var hashedPassword = passwordService.GenerateHash(user.Password);
user.PasswordHash = hashedPassword.PasswordHash; user.PasswordVer =
hashedPassword.PasswordVer; user.PasswordSalt =
hashedPassword.PasswordSalt;
var savedUser = await userService.Create(user); var tokenPayload = new
JwtPayload()
{
UserId = savedUser.Id,
Claims = new List<Claim>()
{
new (JwtRegisteredClaimNames.Email, user.Email)
};
};
var tokens = tokenManager.GenerateTokens(tokenPayload);
await tokenService.Create(tokens.RefreshToken);
var result = new ProfileDto()
{
Token = tokens, User = savedUser
};
return result;
private async Task<bool> ValidateUserEmail(string email) {
await userService.GetUserByEmail(email);
catch (Exception e) when(e is EmailNotFoundException or
EntitiesNotFoundException)

```

```
{  
  return true;  
  return false;  
}  
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Реалізація програмного забезпечення системи графічного моделювання”

Обсяг пояснювальної записки: 77 аркушів.

Дата закінчення роботи: 17 червня 2024 р.

Підпис студента _____